

Sentaurus Device User Guide

Version C-2009.06, June 2009

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2009 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

"This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____. "

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.

Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, Design Compiler, DesignWare, Formality, HDL Analyst, HSIM, HSPICE, Identify, iN-Phase, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synclicity, the Synclicity logo, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSI, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (sm)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.

ARM and AMBA are registered trademarks of ARM Limited.

Saber is a registered trademark of SabreMark Limited Partnership and is used under license.

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

Contents

About This Manual	xxxv
Audience	xxxv
Related Publications	xxxvi
Typographic Conventions.....	xxxvi
Customer Support	xxxvii
Accessing SolvNet.....	xxxvii
Contacting the Synopsys Technical Support Center	xxxvii
Contacting Your Local TCAD Support Team Directly.....	xxxviii
Acknowledgments.....	xxxviii

Part I Getting Started 1

Chapter 1 Overview	3
About Sentaurus Device	3
Creating and Meshing Device Structures	5
Tool Flow.....	6
Starting Sentaurus Device.....	6
From Command Line.....	6
From Sentaurus Workbench	8
Simulation Examples	8
Example: Simple MOSFET Id–Vg Simulation	8
Input Command File	9
File Section	10
Electrode Section.....	11
Physics Section	12
Plot Section	13
Math Section	13
Solve Section.....	14
Simulated Id–Vg Characteristic.....	15
Analysis of 2D Output Data.....	17
Example: Advanced Hydrodynamic Id–Vd Simulation.....	18
Input Command File	18
File Section	21
Main Options	22
Parameter File	22
Listing of mos.par.....	23
Report in Protocol File n3_des.log.....	23

Contents

Electrode Section	23
Main Options	23
Physics Section	24
Main Options	25
Interface Physics	26
Main Options	26
Plot Section	27
CurrentPlot Section	27
Main Options	27
Math Section	28
Example	28
Solve Section	29
Two-dimensional Output Data.	33
Example: Mixed-Mode CMOS Inverter Simulation.	33
Input Command File	34
Device Section	36
System Section.	37
File Section	38
Plot Section	38
Math Section	39
Solve Section	39
Results of Inverter Transient Simulation.	40
Example: Small-Signal AC Extraction	41
Input Command File	41
Device Section	43
File Section	44
System Section.	44
Solve Section	44
Results of AC Simulation	46

Chapter 2 Basic Sentaurus Device	47
Overview	47
Specifying the Device	47
Defining the Output	47
Specifying the Simulation	48
Inserting Files	48
File Section	49
Electrode Section	50
Workfunction and Material Specifications for Contacts	51
Thermode Section	53

Physics Section	54
Physics Section Example.....	54
Region-specific and Material-specific Physics	55
Physics at Interfaces.....	56
Interface Model Syntax	56
Interface Model Parameters	57
Physics at Electrodes	57
Plot Section	58
Interface Plots	58
CurrentPlot Section.....	59
Example: Node Numbers.....	61
Example: Mixed Mode	62
Example: Advanced Options	62
Example: Physical Parameter Values	63
TaileDistributionPlot and TailhDistributionPlot Sections	63
TrappedCarDistrPlot Section	63
NonLocalPlot Section	64
Solve Section.....	64
Coupled Command	65
Plugin Command	66
Quasistationary Command.....	67
Ramping Boundary Conditions	68
Ramping Quasi-Fermi Potentials in Doping Wells	69
Ramping Physical Parameter Values	71
Saving and Plotting Data During Quasistationary Solve Sequence	73
Extrapolation	73
Continuation Command.....	75
Transient Command.....	79
Extrapolation	80
Large-Signal Cyclic Analysis	81
Description of Method	81
Using Cyclic Analysis	83
Plot, Save, and Load Commands.....	84
Example: Solve Section of Input File with Plot, Save, and Load Operations	85
Example: Solve Section of Input File with Multiple Save and Load Operations..	86
System Command	87
NewCurrentPrefix Statement.....	87
CurrentPlot Section	88
Example: CurrentPlot Statements.....	90
Set Command	91

Contents

Math Section	91
Device-specific Math Keywords	92
Physics-related Math Keywords	92
Derivatives	93
Discretization Methods	93
Math Parameters for Transient Analysis	94
Linear Solver-oriented Math Keywords	95
Nonlinear Solver-oriented Math Keywords	97
Keywords for Transient and Quasistationary Control	99
Break Criteria	99
Global Contact Criteria	99
Global Device Criteria	100
Sweep-specific Break Criteria	101
Mixed Mode	102
Parallelization	102
Nonlocal Meshes	104
Specifying Nonlocal Meshes	104
Visualizing Nonlocal Meshes	105
Visualizing Data Defined on Nonlocal Meshes	106
Constructing Nonlocal Meshes	107
Special Handling of 1D Schrödinger Equation	109
Special Handling of Nonlocal Tunneling Model	109
Unnamed Meshes	110
Performance Suggestions	110
Monitoring Convergence Behavior	111
CNormPrint	111
NewtonPlot	112
Incomplete Newton Algorithm	112
Extended Precision	113
Specifying Model Parameters	115
Specifying Region and Material Parameters	115
Generating a Copy of Parameter File	115
Changing Parameter Values in Parameter File	116
Library of Materials	117
Parameters of Compound Materials	119
Undefined Physical Models	119
Default Parameters	120
Hierarchy of Models and Parameters	121
Model Hierarchy	121
Parameter Hierarchy	122

Material and Doping Specification	123
User-Defined Materials	124
User-Defined Species	125
Tcl Command File	127
Overview	127
sdevice Command	130
sdevice_init Command	130
sdevice_solve Command	130
sdevice_finish Command	130
sdevice_parameters Command	131
Flowchart	131
Extraction	132
List of Available Inspect Tcl Commands	132
Output Redirection	134
Known Restrictions	134
Saving Snapshots	134
Extraction File	135
Extraction File Format	135
Analysis Modes	137
File Section	138
Electrode Section	138
Extraction Section	138
Solve Section	139
References	140

Chapter 3 Mixed-Mode Sentaurus Device	141
Overview	141
Compact Models	142
Hierarchical Description of Compact Models	142
Example: Compact Models	145
SPICE Circuit Files	147
Device Section	148
System Section	149
Physical Devices	150
Circuit Devices	151
Electrical and Thermal Netlist	151
Set, Unset, Initialize, and Hint	154
System Plot	155
AC System Plot	155
File Section	156
SPICE Circuit Models	157

Contents

User-Defined Circuit Models	157
Solve Section	158
Coupled Command	158
Circuit and Contact Equation–Variable Keywords	158
Selecting Individual Devices	159
Quasistationary Command	160
ACCoupled: Small-Signal AC Analysis	161
Example: AC Analysis of Simple Device	162
Optical AC Analysis	163
Harmonic Balance	164
Modes of Harmonic Balance Analysis	165
Performing Harmonic Balance Analysis	165
Harmonic Balance Analysis Output	168
Application Notes	169
Set and Unset Section	169
Accessing SPICE Vector Parameters	169
Math Section	170
Using Mixed-Mode Simulation	171
From Single-Device File to Multidevice File	171
File-naming Convention: Mixed-Mode Extension	173
Part II Physics in Sentaurus Device	175
<hr/>	
Chapter 4 Introduction to Physics in Sentaurus Device	177
Transport Equations	177
Poisson Equation and Continuity Equations	178
Drift-Diffusion Model	179
Thermodynamic Transport Model	179
Thermodynamic Model	179
Using the Thermodynamic Model	180
Uniform Self-Heating	182
Using Uniform Self-Heating	183
Hydrodynamic Transport Model	184
Using the Hydrodynamic Model	184
Hydrodynamic Model	184
Hydrodynamic Model Parameters	188
Singlet Exciton Equation	189
Boundary Conditions for Singlet Exciton Equation	190
Using the Singlet Exciton Equation	191
Conductivity of Metals	193
Conductive Insulators	195

Current Potential	199
Quasi-Fermi Potential	201
Fermi Statistics	202
Using Fermi Statistics	203
Multivalley Statistics	203
Using Multivalley Statistics	204
Boundary Conditions	206
Electrical Boundary Conditions	206
Ohmic Contacts	206
Gate Contacts	207
Schottky Contacts	208
Barrier Lowering at Schottky Contacts	209
Resistive Contacts	210
Floating Metal Gates	211
Floating Semiconductor Gates	213
Boundaries Without Contacts	214
Thermal Boundary Conditions for Thermodynamic Model	215
Thermal Boundary Conditions for Hydrodynamic Model	215
Total Thermal Resistance	216
Estimating Thermal Resistance	217
Periodic Boundary Conditions	217
Starting Solution or ‘Initial Guess’	219
Electrostatic Potential and Quasi-Fermi Potentials in Doping Wells	219
Regionwise Specification of Initial Quasi-Fermi Potentials	220
Thermodynamic and Hydrodynamic Simulations	220
Save File Overrides the Initial Guess	220
References	221
Chapter 5 Semiconductor Band Structure	225
Intrinsic Density	225
Band Gap and Electron Affinity	225
Selecting the Bandgap Model	226
Bandgap and Electron-Affinity Models	226
Bandgap Narrowing for Bennett–Wilson Model	227
Bandgap Narrowing for Slotboom Model	228
Bandgap Narrowing for del Alamo Model	228
Bandgap Narrowing for Jain–Roulston Model	228
Table Specification of Bandgap Narrowing	230
Schenk Bandgap Narrowing Model	231
Bandgap Narrowing with Fermi Statistics	235
Bandgap Parameters	236

Contents

Effective Masses and Effective Density-of-States	237
Electron Effective Mass and DOS.....	237
Formula 1	237
Formula 2	238
Electron Effective Mass and Conduction Band DOS Parameters.....	238
Hole Effective Mass and DOS.....	239
Formula 1	239
Formula 2	239
Hole Effective Mass and Valence Band DOS Parameters	240
Gaussian Density-of-States for Organic Semiconductors	240
References.....	242
<hr/>	
Chapter 6 Incomplete Ionization	245
Overview.....	245
Using Incomplete Ionization.....	246
Incomplete Ionization Model	246
Physical Model Parameters.....	248
References.....	249
<hr/>	
Chapter 7 Quantization Models	251
Overview.....	251
van Dort Quantization Model	252
van Dort Model	252
Using the van Dort Model	253
1D Schrödinger Solver	253
Nonlocal Mesh for 1D Schrödinger.....	254
Using 1D Schrödinger	255
1D Schrödinger Parameters	255
Explicit Ladder Specification.....	256
Automatic Extraction of Ladder Parameters	256
Visualizing Schrödinger Solutions	258
1D Schrödinger Model.....	258
1D Schrödinger Application Notes	259
Density Gradient Quantization Model	260
Density Gradient Model.....	260
Using the Density Gradient Model	261
Density Gradient Application Notes	263
Modified Local-Density Approximation	264
MLDA Model	264
Using MLDA.....	264

MLDA Application Notes	266
References.....	266

Chapter 8 Mobility Models	267
How Mobility Models Combine	267
Mobility due to Phonon Scattering	268
Doping-dependent Mobility Degradation.....	268
Using Doping-dependent Mobility	269
Masetti Model	269
Arora Model.....	270
University of Bologna Bulk Mobility Model	271
PMIs for Bulk Mobility	273
Mobility Degradation at Interfaces	273
Using Mobility Degradation at Interfaces	273
Enhanced Lombardi Model	274
Enhanced Lombardi Model with High-k Degradation	276
Inversion and Accumulation Layer Mobility Model.....	279
Using Inversion and Accumulation Layer Mobility Model.....	283
University of Bologna Surface Mobility Model	284
Computing Transverse Field	286
Normal to Interface.....	286
Normal to Current Flow	287
Field Correction on Interface	287
Carrier–Carrier Scattering	288
Using Carrier–Carrier Scattering.....	288
Conwell–Weisskopf Model	288
Brooks–Herring Model	289
Physical Model Parameters	289
Philips Unified Mobility Model	289
Using Philips Model	290
Using an Alternative Philips Model.....	290
Philips Model Description	291
Screening Parameter	292
Philips Model Parameters	293
High-Field Saturation	294
Using High-Field Saturation	294
Extended Canali Model	295
Transferred Electron Model.....	297
Basic Model.....	297
Meinerzhagen–Engl Model	298
Lucent Model.....	298

Contents

Velocity Saturation Models	299
Selecting Velocity Saturation Models	299
Driving Force Models	300
Monte Carlo-computed Mobility for Strained Silicon	301
Monte Carlo-computed Mobility for Strained SiGe in npn-SiGe HBTs	302
Incomplete Ionization-dependent Mobility Models	302
Poole-Frenkel Mobility (Organic Material Mobility)	303
Mobility Averaging	304
Mobility Doping File	304
References	305
<hr/>	
Chapter 9 Generation–Recombination	309
Shockley–Read–Hall Recombination	309
Using SRH Recombination	310
SRH Doping Dependence	311
Lifetime Profiles from Files	311
SRH Temperature Dependence	312
SRH Doping- and Temperature-dependent Parameters	313
SRH Field Enhancement	313
Using Field Enhancement	314
Schenk Trap-assisted Tunneling (TAT) Model	314
Schenk TAT Density Correction	316
Hurkx TAT Model	316
Field-Enhancement Parameters	317
Trap-assisted Auger Recombination	318
Surface SRH Recombination	319
Coupled Defect Level (CDL) Recombination	320
Using CDL	320
CDL Model	320
Radiative Recombination	321
Using Radiative Recombination	321
Radiative Model	322
Auger Recombination	322
Avalanche Generation	324
Using Avalanche Generation	324
van Overstraeten – de Man Model	325
Okuto–Crowell Model	326
Lackner Model	326
University of Bologna Impact Ionization Model	327
New University of Bologna Impact Ionization Model	329
Driving Force	331

Avalanche Generation with Hydrodynamic Transport	331
Approximate Breakdown Analysis: Poisson Equation Approach	333
Using Breakdown Analysis	333
Band-to-Band Tunneling Models	335
Using Band-to-Band Tunneling.....	335
Schenk Model	336
Schenk Density Correction.....	337
Simple Band-to-Band Models	337
Hurkx Band-to-Band Model	338
Dynamic Nonlocal Path Band-to-Band Model	339
Band-to-Band Generation Rate	339
Using Nonlocal Path Band-to-Band Model	342
Visualizing Nonlocal Band-to-Band Generation Rate	343
Tunneling Near Interfaces and Equilibrium Regions	344
Bimolecular Recombination	344
Physical Model	344
Using Bimolecular Recombination	345
References.....	345

Chapter 10 Traps and Fixed Charges	349
Basic Syntax for Traps	349
Trap Types	350
Energetic and Spatial Distribution of Traps	350
Trap Models and Parameters.....	352
Trap Occupation Dynamics.....	352
Local Trap Capture and Emission	354
J-Model Cross Sections	355
Hurkx Model for Cross Sections	355
Poole–Frenkel Model for Cross Sections.....	355
Local Capture and Emission Rates Based on Makram-Ebeid–Lannoo Phonon-assisted Tunnel Ionization Model	356
Local Capture and Emission Rates from PMI	357
Tunneling and Traps	357
Trap Numeric Parameters	359
Visualizing Traps	359
Explicit Trap Occupation	361
Trap Examples	362
Insulator Fixed Charges	363
References.....	364

Contents

Chapter 11 Phase and State Transitions	365
Multistate Configurations and Their Dynamic	365
Specifying Multistate Configurations	367
Interaction of Multistate Configurations with Transport	368
Apparent Band-Edge Shift	368
Thermal Conductivity, Heat Capacity, and Mobility	369
Manipulating MSCs During Solve	369
Explicit State Occupations	370
Manipulating Transition Dynamics	371
Example: Two-State Phase-Change Memory Model	371
Chapter 12 Degradation Model	373
Overview	373
Trap Degradation Model	373
Trap Formation Kinetics	373
Power Law and Kinetic Equation	374
Si-H Density-dependent Activation Energy	374
Diffusion of Hydrogen in Oxide	375
Syntax and Parameterized Equations	376
Device Lifetime and Simulation	378
Hydrogen Transport Degradation Model	381
Hydrogen Transport	381
Reactions Between Mobile Elements	381
Reactions with Multistate Configurations	384
Using Hydrogen Transport Degradation Model	385
References	386
Chapter 13 Organic Devices	389
Introduction to Organic Device Simulation	389
References	390
Chapter 14 Optical Generation	393
Unified Interface for Optical Generation Computation	393
Specifying the Type of Optical Generation Computation	394
Optical Generation from Monochromatic Source	395
Illumination Spectrum	396
Loading and Saving Optical Generation from and to File	397
Constant Optical Generation	397

Specifying Time Dependency for Transient Simulations.....	398
Solving the Optical Problem.....	400
Parameter Ramping	406
Raytracing	407
Raytracer	407
Ray Photon Absorption and Optical Generation	410
Using the Raytracer	410
Monte Carlo Raytracing.....	412
Multithreading for Raytracer	413
Compact Memory Model for Raytracer.....	414
Window of Starting Rays.....	414
Two-dimensional Device and Window Description	415
Three-dimensional Device and Window Description	416
Spatial Distribution of Intensity	418
Boundary Condition for Raytracing	422
Constant Reflectivity and Transmittivity Boundary Condition	423
Raytrace PMI Boundary Condition	424
Thin-Layer-Stack Boundary Condition	425
TMM Optical Generation in Raytracer	426
Virtual Regions in Raytracer	427
Additional Options for Raytracing	427
Optical Generation Scaling for Raytracing	428
Redistributing Power of Stopped Rays	428
Visualizing Raytracing	429
Reporting Various Powers in Raytracing	429
Dual-Grid Setup for Raytracing.....	430
Old Raytracer.....	432
Polarization of Old Raytracer.....	434
Optical Beam Absorption	434
Physical Model	435
Using Optical Beam Absorption	436
Transfer Matrix Method	438
Physical Model	438
Using Transfer Matrix Method	442
Finite-Difference Time-Domain Method	447
Files of EMW Generation	447
Creating the Tensor Grid and EMW Input Command File	447
User-Defined Input or Tensor Grid	448
Log of EMW Run.....	449
Syntax of EMW Generation: Input File of EMW	449
Boundary	450

Contents

Excitation	451
Excitation Example.....	453
Material/AutoMatGen	453
EMW Generation: Tensor Grid, Syntax, and Algorithm.....	456
GridBoundX, GridBoundY, and GridBoundZ.....	457
SmoothingFactor	459
Multithreading for EMW Generation: Syntax.....	460
EMW Interface to Hardware Acceleration	460
Beam Propagation Method	462
Physical Model	462
Bidirectional BPM	463
Boundary Conditions	463
Using Beam Propagation Method	464
General	464
Bidirectional BPM	465
Excitation	466
Boundary	469
Optics Stand-Alone Option	470
Ramping Input Parameters	470
Visualizing Results on Native Tensor Grid	471
Refractive Index Models.....	472
Absorption Models	473
Default Absorption Model from Parameter File	473
Table-based Optical Properties of Materials in Parameter File.....	475
Absorption Coefficient Model.....	475
Complex Refractive Index Model.....	477
Physical Model	477
Wavelength Dependency	478
Temperature Dependency.....	478
Carrier Dependency	479
Gain Dependency	479
Using Complex Refractive Index	480
Loading Optical Generation from File	483
Wavelength Ramping for Computing Optical Generation	484
Spectral Illumination.....	486
Optical AC Analysis	487
References.....	488
Chapter 15 Radiation Models	489
Generation by Gamma Radiation	489
Using Gamma Radiation Model	489

Yield Function	490
Alpha Particles	490
Using Alpha Particle Model	490
Alpha Particle Model.....	491
Heavy Ions	492
Using Heavy Ion Model.....	492
Heavy Ion Model.....	493
Examples: Heavy Ions.....	495
Example 1.....	495
Example 2.....	496
Example 3.....	496
Improved Alpha Particle/Heavy Ion Generation Rate Integration	497
References.....	498

Chapter 16 Noise and Fluctuation Analysis 499

Performing Noise and Fluctuation Analysis	499
Noise Sources	501
Diffusion Noise	501
Equivalent Monopolar Generation–Recombination Noise	502
Bulk Flicker Noise.....	502
Trapping Noise	503
Random Dopant Fluctuations	503
Noise from SPICE Circuit Elements	504
Impedance Field Method	504
Noise Output Data.....	506
References.....	509

Chapter 17 Tunneling 511

Tunneling Model Overview	511
Fowler–Nordheim Tunneling	512
Using Fowler–Nordheim	512
Fowler–Nordheim Model	513
Fowler–Nordheim Parameters.....	514
Direct Tunneling	514
Using Direct Tunneling	514
Direct Tunneling Model.....	515
Image Force Effect	516
Direct Tunneling Parameters	517
Nonlocal Tunneling at Interfaces, Contacts, and Junctions	518
Defining Nonlocal Meshes	518

Contents

Specifying Nonlocal Tunneling Model	520
Nonlocal Tunneling Parameters.	521
Visualizing Nonlocal Tunneling	523
Physics of Nonlocal Tunneling Model	523
WKB Tunneling Probability.	524
Schrödinger Equation-based Tunneling Probability	526
Density Gradient Quantization Correction	526
Nonlocal Tunneling Current	527
Band-to-Band Contributions to Nonlocal Tunneling Current	527
Carrier Heating	528
References	529
<hr/>	
Chapter 18 Hot-Carrier Injection Models	531
Overview	531
Destination of Injected Current	531
Injection Barrier and Image Potential	534
Effective Field	535
Classical Lucky Electron Injection	535
Fiegna Hot-Carrier Injection	536
Tail Distribution Hot-Carrier Injection	537
Tail Distribution Model	538
Injection Model	540
Using Tail Distribution Hot-Carrier Injection Model	541
Visualizing Tail Distribution	544
Carrier Injection with Explicitly Evaluated Boundary Conditions for Continuity Equations	
545	
References	546
<hr/>	
Chapter 19 Heterostructure Device Simulation	547
Physics Models and Differential Equations	547
Mole-Fraction Materials	548
Mole-Fraction Specification	550
Composition-dependent Models	551
Ternary Semiconductor Composition	552
Example 1: Specifying Electron Effective Mass	553
Example 2: Specifying Electric Permittivity	554
Example 3: Specifying Band Gap	554
Quaternary Semiconductor Composition	556
Default Model Parameters for Compound Semiconductors	557
Abrupt and Graded Heterojunctions	558

Thermionic Emission Current.....	559
Using Thermionic Emission Current.....	559
Thermionic Emission Model.....	560
Gaussian Transport Across Organic Heterointerfaces	561
Using Gaussian Transport at Organic Heterointerfaces	561
Gaussian Transport at Organic Heterointerface Model.....	562
References.....	562

Chapter 20 Energy-dependent Parameters 565

Overview.....	565
Energy-dependent Energy Relaxation Time.....	565
Spline Interpolation	567
Energy-dependent Mobility	568
Spline Interpolation	570
Energy-dependent Peltier Coefficient.....	571
Spline Interpolation	572

Chapter 21 Anisotropic Properties 575

Overview.....	575
Anisotropic Mobility.....	576
Crystal Reference System	576
Anisotropy Factor	576
Current Densities	577
Driving Forces	578
Total Anisotropic Mobility	580
Total Direction-dependent Anisotropic Mobility	580
Self-Consistent Anisotropic Mobility	581
Plot Section	582
Anisotropic Avalanche Generation.....	583
Anisotropic Electrical Permittivity	584
Anisotropic Thermal Conductivity	586
Anisotropic Density Gradient Model	587

Chapter 22 Ferroelectric Materials 589

Using Ferroelectrics	589
Ferroelectrics Model	591
References.....	593

Chapter 23 Modeling Mechanical Stress Effect	595
Overview.....	595
Deformation of Band Structure.....	596
Using Deformation Potential Model	599
Strained Effective Masses and Density-of-States	601
Strained Electron Effective Mass and DOS.....	601
Strained Hole Effective Mass and DOS	603
Using Strained Effective Masses and DOS.....	604
Multivalley Band Structure.....	605
Using Multivalley Band Structure.....	605
Piezoresistance Mobility Model	606
Using Piezoresistance Mobility Model	608
Enormal- and MoleFraction-dependent Piezo Coefficients	609
Using Piezoresistive Prefactors Model	609
Example 1.....	610
Example 2.....	611
Example 3.....	613
Piezoresistance Mobility Factor Models.....	615
Using Piezoresistance Mobility Factor Models.....	616
Stress-induced Electron Mobility Model	617
Intervalley Scattering.....	620
Effective Mass	621
Using Stress-induced Electron Mobility Model	622
Intel Stress-induced Hole Mobility Model	624
Stress Dependencies	626
Generalization of Model	626
Using Intel Mobility Model.....	628
Using Stress-dependent Models	629
Tensor Grid Option	630
Mobility Enhancement Limits	631
Plotting Mobility Enhancement Factors	631
Stress Mobility Model for Minority Carriers	632
Dependency of Saturation Velocity on Stress.....	633
Stress Tensor Applied to Low-Field Mobility	634
Piezoelectric Polarization	635
Strain Model	636
Stress Model	636
Poisson Equation	637
Parameter File	637

Coordinate Systems	638
References.....	639
<hr/>	
Chapter 24 Galvanic Transport Model	643
Model Description.....	643
Using Galvanic Transport Model	644
Numeric Computation of Current Vector Without Mobility	644
References.....	645
<hr/>	
Chapter 25 Thermal Properties	647
Heat Capacity	647
Thermal Conductivity.....	648
Thermoelectric Power (TEP)	649
References.....	650
Part III Physics of Lasers and Light-Emitting Diodes	653
<hr/>	
Chapter 26 Introduction to Lasers and Light-emitting Diodes	655
Overview.....	656
Command File Syntax.....	657
Simulating Single-Grid Edge-emitting Laser	658
Simulating Dual-Grid Edge-emitting Laser.....	662
Simulating Vertical-Cavity Surface-emitting Laser	666
Simulating Light-emitting Diode.....	671
Simulating Organic Light-emitting Diode.....	676
Investigating Simulation Results	680
Current File and Plot Variables for Laser Simulation	681
Current File and Plot Variables for LED Simulation	683
Plot Variables for Quantum-Well Modeling	685
<hr/>	
Chapter 27 Lasers	687
Overview.....	687
Coupling Between Optics and Electronics	688
Algorithm for Coupling Electrical and Optical Problems.....	689
Photon Rate and Photon Phase Equations.....	690
Laser Small-Signal AC Analysis and Modulation Response	693
Intensity and Photon Phase Modulation Response	693
Syntax for Small-Signal AC Extraction.....	694

Contents

Modeling Relative Intensity Noise and Frequency Noise	695
Relative Intensity Noise and Frequency Noise	695
Small-Signal Noise Modeling	696
Syntax for Relative Intensity Noise and Frequency Noise Model.....	697
Plotting the Laser Power Spectrum.....	698
Refractive Index, Dispersion, and Optical Loss	699
Temperature Dependence of Refractive Index	699
Carrier-Density Dependence of Refractive Index	700
Wavelength Dependence and Absorption of Refractive Index	701
Free Carrier Loss.....	702
Edge-emitting Lasers	703
Waveguide Optical Modes and Fabry–Perot Cavity	703
Lasing Wavelength in Fabry–Perot Cavity	705
Output Power from a Fabry–Perot Laser.....	705
Symmetry Considerations	706
Multiple Transverse Modes.....	706
Multiple Longitudinal Modes	707
Specifying Fixed Optical Confinement Factor	708
Simple Distributed Feedback Model	708
Bulk Active-Region Edge-emitting Lasers	709
Leaky Waveguide Lasers.....	709
Device Physics and Parameter Tuning	710
Vertical-Cavity Surface-emitting Lasers	711
Cavity Optical Modes in VCSELs.....	711
VCSEL Output Power	714
Cylindrical Symmetry	715
Different Grid and Structure for Electrical and Optical Problems	716
Aligning Resonant Wavelength Within Gain Spectrum	717
Approximate Methods for VCSEL Cavity Problem	718
Device Physics and Parameter Tuning	718
References.....	719

Chapter 28 Light-emitting Diodes	721
Modeling Light-emitting Diodes	721
Coupling Electronics and Optics in LED Simulations	722
Discussion of LED Physics.....	723
LED Raytracing	724
Single-Grid Versus Dual-Grid LED Simulation	725
Isotropic Starting Rays from Spontaneous Emission Sources.....	725
Anisotropic Starting Rays from Spontaneous Emission Sources	726
Randomizing Starting Rays	727

Reading Starting Rays from File	728
Moving Starting Rays on Boundaries	728
Debugging Raytracing	729
Print Options in Raytracing	730
LED Radiation Pattern	731
Two-dimensional LED Radiation Pattern and Output Files	733
Three-dimensional LED Radiation Pattern and Output Files	734
Spectrum-dependent LED Radiation Pattern	735
Tracing Source of Output Rays	736
Nonactive Region Absorption (Photon Recycling)	736
Accelerating Gain Calculations and LED Simulations	737
Active-Region Photon Recycling	738
Simple Photon-Recycling Model (Active Region)	738
Setting Up Simple Photon-Recycling Model	740
Defining Photon-Recycling Contacts	740
Specifying Photon-Recycling Coefficients	742
Iterating Active-Region Photon-Recycling Effect Self-consistently	743
Full Photon-Recycling Model (Active Region)	743
Setting Up Full Photon-Recycling Model	746
Syntax for Full Photon-Recycling Model	746
Plotting Evolving Spectra	748
Spectral Conversion	749
Defining Absorption and Emission Profiles	750
Spectral Conversion Syntax	752
Henyey–Greenstein Volume Scattering Probability	753
Interface to <i>LightTools®</i>	754
Example: orainterface_rays.txt	756
Example: orainterface_spectrum.txt	756
Device Physics and Tuning Parameters	756
Modeling Organic Light-emitting Diodes	757
OLED Emissions	758
Organic Material Parameters	758
References	760
Chapter 29 Optical Mode Solver for Lasers	761
Overview	761
Finite-Element Formulation	762
Syntax of FE Scalar and FE Vectorial Optical Solvers	763
FE Scalar Solver	764
FE Vectorial Solver	765
FE Vectorial Solver for Waveguides	765

Contents

FE Vectorial Solver for VCSEL Cavity	766
Specifying Multiple Entries for Parameters in FEScalar and FEVectorial	767
Multiple Waveguide Modes	768
Multiple Cavity Modes	768
Boundary Conditions and Symmetry for Optical Solvers	769
Symmetric FEScalar Waveguide Mode in Cartesian Coordinates	770
Symmetric FEVectorial Waveguide Modes in Cartesian Coordinates	771
Symmetric FEVectorial VCSEL Cavity Modes in Cartesian Coordinates	771
Symmetric FEVectorial VCSEL Cavity Modes in Cylindrical Coordinates	772
Perfectly Matched Layers	773
Transfer Matrix Method for VCSELs	775
Effective Index Method for VCSELs	777
Formulation of Effective Index Method	778
Transverse Mode Pattern of VCSELs	780
Syntax for Effective Index Method	781
Cylindrical Modes	781
Cartesian Modes	782
Saving and Loading Optical Modes	783
Saving Optical Modes on Optical or Electrical Mesh	783
Loading Optical Modes	784
Loading Optical Fields for Edge-emitting Laser	785
Loading Optical Fields for VCSEL	785
Far Field	786
Far-Field Observation Angle	788
Syntax for Far Field	789
Far-Field Output Files	790
Scalar1D Far Field	790
Scalar2D and Vector2D Far Fields	791
Far Field from Loaded Optical Field File	793
VCSEL Near Field and Far Field	793
Optics Stand-alone Option	795
Automatic Optical Mode Searching	797
Syntax for Automatic Mode Searching	798
Searching for Cavity Resonances	799
Searching for Waveguide Modes	800
References	801
Chapter 30 Modeling Quantum Wells	803
Overview	803
Carrier Capture in Quantum Wells	804
Special Meshing Requirements for Quantum Wells	805

Thermionic Emission	805
Quantum-Well Scattering Model	806
Radiative Recombination and Gain Coefficients	809
Stimulated and Spontaneous Emission Coefficients	809
Active Bulk Material Gain	811
Stimulated Recombination Rate	811
Spontaneous Recombination Rate	812
Spontaneous Emission Power for LEDs	812
Fitting Stimulated and Spontaneous Emission Spectra	813
Gain-broadening Models	814
Lorentzian Broadening	814
Landsberg Broadening	814
Hyperbolic-Cosine Broadening	815
Syntax to Activate Broadening	815
Nonlinear Gain Saturation Effects	815
Simple Quantum-Well Subband Model	817
Syntax for Simple Quantum-Well Model	820
Strain Effects	820
Syntax for Quantum-Well Strain	821
Polarization-dependent Optical Matrix Element	822
The k.p Method	825
Luttinger-Kohn Parameters and Hamiltonians for Zinc-Blende Crystal Structure ..	827
Four-Band Hamiltonian	828
Six-Band Hamiltonian	828
Eight-Band Hamiltonian	829
Luttinger-Kohn Parameters and Hamiltonian for Wurtzite Crystal Structure ..	830
Six-Band Hamiltonian	830
Syntax for k.p Method	831
Constructing Nonlocal Mesh on Straight Line	831
Adjusting k.p Parameters in Parameter File	836
Plotting the Local Band Structure Data	838
Screening Piezoelectric Fields in GaN-type Quantum Wells	839
Optical Emission Including Manybody Effects	840
Physical Model	840
Free Carrier Theory	843
Screened Hartree-Fock Approximation	843
Second Born Approximation	844
Computing Tabulated Optical Emission Data	844
Computing Emission Tables	844
Computing Dephasing Rates	847
Loading Tabulated Stimulated and Spontaneous Emission	848

Contents

Importing Gain and Spontaneous Emission Data with PMI	849
Implementing Gain PMI	849
References	852
<hr/>	
Chapter 31 Additional Features of Laser or LED Simulations	855
Plotting Gain and Power Spectra	855
Modal Gain as Function of Bias	855
Material Gain in Active Region	855
Modal Gain as Function of Energy/Wavelength	856
Transient Simulation	857
Syntax for Laser Transient Simulation	857
Performing Temperature Simulation	859
Lattice Temperature Simulation	860
Carrier Temperature Simulation	862
Switching from Voltage to Current Ramping	863
Robust Wavelength Search Algorithm	865
Reducing Number of Result Files	865
Scripts	866
Part IV Mesh and Numeric Methods	
867	
<hr/>	
Chapter 32 Automatic Grid Generation and Adaptation Module AGM	869
Overview	869
Adaptation Procedure	870
Adaptation Decision	871
Adaptation Strategy	871
Adaptation Criteria	872
Criteria Based on Local Dirichlet Problems	872
Residual Adaptation Criteria	873
Criteria Based on Element Variation	873
Solution Recomputation	874
Device-Level Data Smoothing	874
System-Level Data Smoothing	874
Adaptive Device Instances	875
AGM Device Parameters	875
Parameters Affecting Grid Generation	875
Device Parameters Affecting Smoothing	876
Grid Specification	877
Adaptation Criteria	878
General	878

Dirichlet	878
Residual	879
Element	879
Adaptive Solve Statements	879
General Adaptive Solve Statements	879
Adaptive Coupled Solve Statements	880
Adaptive Quasistationary Solve Statements	880
Limitations and Recommendations	881
Limitations	881
Recommendations	881
Accuracy of Terminal Currents as Adaptation Goal	881
AGM Simulation Times	881
Dirichlet Versus Residual Adaptation Criteria	882
Large Grid Sizes	882
Convergence Problems After Adaptation	883
AGM and Extrapolate	883
References	883
<hr/>	
Chapter 33 Numeric Methods	885
Discretization	885
Box Method Coefficients	886
Basic Definitions	886
Variation of Box Method Algorithms	888
Truncated and Non-Delaunay Elements	889
Math Parameters for Box Method Coefficients	890
Saving and Restoring Box Method Coefficients	891
AC Simulation	893
AC Response	893
AC Current Density Responses	895
Harmonic Balance Analysis	895
Harmonic Balance Equation	896
Multitone Harmonic Balance Analysis	896
Multidimensional Fourier Transformation	896
Quasi-Periodic Functions	897
Multidimensional Frequency Domain Problem	898
One-Tone Harmonic Balance Analysis	898
Solving HB Equation	899
Solving HB Newton Step Equation	900
Restarted GMRES Method	900
Direct Solver Method	901

Contents

Transient Simulation	901
Backward Euler Method	901
TRBDF Composite Method	902
Controlling Transient Simulations	903
Nonlinear Solvers	904
Fully Coupled Solution	904
‘Plugin’ Iterations	906
References	907

Part V Physical Model Interface	909
--	------------

Chapter 34 Physical Model Interface	911
Overview	911
C++ Interface	912
Shared Object Code	916
Command File of Sentaurus Device	916
Vertex-based Run-Time Support	918
Vertex-based Run-Time Support for Multistate Configuration-dependent Models	922
Mesh-based Run-Time Support	923
Device Mesh	924
Vertex	924
Edge	925
Element	925
Region	927
Region Interface	928
Mesh	929
Device Data	930
Parameter File of Sentaurus Device	932
Generation–Recombination Model	933
Dependencies	933
C++ Interface	934
Example: Auger Recombination	935
Avalanche Generation Model	935
Dependencies	936
C++ Interface	937
Example: Okuto Model	938
Mobility Models	942
Doping-dependent Mobility	942
Dependencies	942
C++ Interface	943
Example: Masetti Model	944

Multistate Configuration-dependent Bulk Mobility	947
Command File	947
Dependencies	948
C++ Interface	948
Mobility Degradation at Interfaces	949
Dependencies	949
C++ Interface	951
Example: Lombardi Model	952
High-Field Saturation Model	957
Dependencies	957
C++ Interface	958
Example: Canali Model	960
Band Gap	965
Dependencies	966
C++ Interface	966
Example: Default Bandgap Model	967
Bandgap Narrowing	968
Dependencies	968
C++ Interface	969
Example: Default Model	969
Apparent Band-Edge Shift	970
Dependencies	971
C++ Interface	972
Multistate Configuration-dependent Apparent Band-Edge Shift	973
Dependencies	973
Additional Functionality	974
Using Dependencies	974
Updating Actual Status	974
C++ Interface	974
Electron Affinity	975
Dependencies	976
C++ Interface	976
Example: Default Affinity Model	977
Effective Mass	978
Dependencies	978
C++ Interface	979
Example: Linear Effective Mass Model	979
Energy Relaxation Times	981
Dependencies	982
C++ Interface	982
Example: Constant Energy Relaxation Times	983

Contents

Lifetimes	985
Dependencies	985
C++ Interface	985
Example: Doping- and Temperature-dependent Lifetimes	986
Thermal Conductivity	989
Dependencies	989
C++ Interface	989
Example: Temperature-dependent Thermal Conductivity	990
Multistate Configuration-dependent Thermal Conductivity	991
Command File	991
Dependencies	992
C++ Interface	993
Heat Capacity	994
Dependencies	994
C++ Interface	995
Example: Constant Heat Capacity	995
Multistate Configuration-dependent Heat Capacity	996
Command File	996
Dependencies	997
C++ Interface	998
Optical Absorption	998
Dependencies	999
C++ Interface	999
Example: Temperature-dependent Absorption Model	999
Refractive Index	1000
Dependencies	1001
C++ Interface	1001
Example: Temperature-dependent Refractive Index	1001
Stress	1002
Dependencies	1003
C++ Interface	1003
Example: Constant Stress Model	1004
Trap Space Factor	1006
Dependencies	1006
C++ Interface	1006
Example: PMI User Field as Space Factor	1007
Trap Capture and Emission Rates	1008
Traps	1008
Multistate Configurations	1008
Dependencies	1008
C++ Interface	1009

Example: CEModel_ArrheniusLaw	1011
Trap Energy Shift	1011
Command File	1011
Dependencies	1012
C++ Interface	1012
Piezoelectric Polarization	1013
Dependencies	1013
C++ Interface	1014
Example: Gaussian Polarization Model	1014
Incomplete Ionization	1015
Dependencies	1016
C++ Interface	1016
Example: Matsuura Incomplete Ionization Model	1017
Hot-Carrier Injection	1022
Dependencies	1022
C++ Interface	1023
Example: Lucky Model	1023
Piezoresistive Coefficients	1026
Dependencies	1026
C++ Interface	1027
Current Plot File of Sentaurus Device	1027
Structure of Current Plot File	1028
C++ Interface	1028
Example: Average Electrostatic Potential	1029
Postprocess for Transient Simulation	1031
C++ Interface	1032
Example: Postprocess User-Field	1032
Special Contact PMI for Raytracing	1033
Dependencies	1034
C++ Interface	1036
Example: Assessing and Modifying a Ray	1037
Spatial Distribution Function	1039
Dependencies	1040
C++ Interface	1040
Example: Gaussian Spatial Distribution Function	1041
References	1042

Contents

Part VI Appendices	1043
<hr/>	
Appendix A Mathematical Symbols	1045
<hr/>	
Appendix B Syntax	1049
<hr/>	
Appendix C File-naming Convention	1051
Overview.....	1051
Compatibility with Old File-naming Convention.....	1052
<hr/>	
Appendix D Command-Line Options	1053
Starting Sentaurus Device.....	1053
Command-Line Options	1053
<hr/>	
Appendix E Run-Time Statistics	1057
The sdevicestat Command	1057
<hr/>	
Appendix F Data and Plot Names	1059
Overview.....	1059
Scalar Data	1060
Vector Data	1084
Special Vector Data	1085
<hr/>	
Appendix G Command File Overview	1087
Organization of Command File Overview	1087
Top Level of Command File.....	1089
Device	1089
File	1090
Solve	1094
System	1108
Boundary Conditions	1110
Math	1112
Physics	1125
Generation and Recombination	1131
Laser and LED.....	1148
Mobility	1161

Contents

Radiation Models.....	1163
Various.....	1164
Plotting	1180
Various	1183

Contents

About This Manual

Sentaurus Device is a fully featured 2D and 3D device simulator that provides you with the ability to simulate a broad range of devices.

Sentaurus Device is a multidimensional, 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 manual is divided into parts:

- Part I contains information about how to start Sentaurus Device and how it interacts with other Synopsys tools.
- Part II describes the physics in Sentaurus Device.
- Part III describes the physical models used in laser and light-emitting diode simulations.
- Part IV presents the automatic grid generation facility and provides background information on the numeric methods used in Sentaurus Device.
- Part V describes the physical model interface, which provides direct access to certain models in the semiconductor transport equations and the numeric methods in Sentaurus Device.
- Part VI contains the appendices.

Audience

This manual is intended for users of the Sentaurus Device software package. It assumes familiarity with working on UNIX-like systems and requires a basic understanding of semiconductor device physics and its terminology.

Related Publications

For additional information about Sentaurus Device, see:

- The TCAD Sentaurus release notes, available on SolvNet (see [Accessing SolvNet on page xxxvii](#)).
 - Documentation on the Web, which is available through SolvNet at <https://solvnet.synopsys.com/DocsOnWeb>.
-

Typographic Conventions

Convention	Explanation
<>	Angle brackets
{ }	Braces
[]	Brackets
()	Parentheses
Blue text	Identifies a cross-reference (only on the screen).
Bold text	Identifies a selectable icon, button, menu, or tab. It also indicates the name of a field, window, dialog box, or panel.
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.
<i>Italicized text</i>	Used for emphasis, the titles of books and journals, and non-English words. It also identifies components of an equation or a formula, a placeholder, or an identifier.
Menu > Command	Indicates a menu command, for example, File > New (from the File menu, select New).
NOTE	Identifies important information.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

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

To access SolvNet:

1. Go to the SolvNet Web page at <https://solvnet.synopsys.com>.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)

If you need help using SolvNet, click Help on the SolvNet menu bar.

Contacting the Synopsys Technical Support Center

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

- Open a call to your local support center from the Web by going to <https://solvnet.synopsys.com/EnterACall> (Synopsys user name and password required).
- Send an e-mail message to your local support center:
 - E-mail support_center@synopsys.com from within North America.
 - Find other local support center e-mail addresses at <http://www.synopsys.com/Support/GlobalSupportCenters/Pages/default.aspx>.
- Telephone your local support center:
 - Call (800) 245-8005 from within the continental United States.
 - Call (650) 584-4200 from Canada.
 - Find other local support center telephone numbers at <http://www.synopsys.com/Support/GlobalSupportCenters/Pages/default.aspx>.

Contacting Your Local TCAD Support Team Directly

Send an e-mail 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.

The third-party software ARPACK (ARnoldi PACKage) by R. Lehoucq, K. Maschhoff, D. Sorensen, and C. Yang is used in Sentaurus Device (<http://www.caam.rice.edu/software/ARPACK>).

Sentaurus Device contains the QD library for double-double and quad-double floating-point arithmetic (<http://crd.lbl.gov/~dhbailey/mpdist>). The QD library requires the following copyright notice:

This work was supported by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098.

Copyright © 2003, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from U.S. Dept. of Energy)

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

(2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

(3) Neither the name of Lawrence Berkeley National Laboratory, U.S. Dept. of Energy nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

About This Manual

Acknowledgments

Part I Getting Started

This part of the Sentaurus Device manual contains the following chapters:

[Chapter 1 Overview on page 3](#)

[Chapter 2 Basic Sentaurus Device on page 47](#)

[Chapter 3 Mixed-Mode Sentaurus Device on page 141](#)

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

About Sentaurus Device

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 on page 4](#) 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 μm .

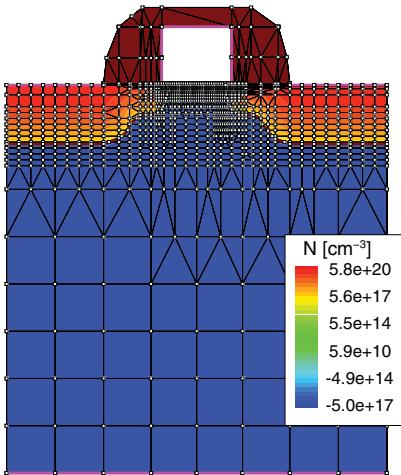


Figure 1 Two-dimensional doping profile that is discretized on the nodes of simulation grid

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

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.

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 on page 5](#)).

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.

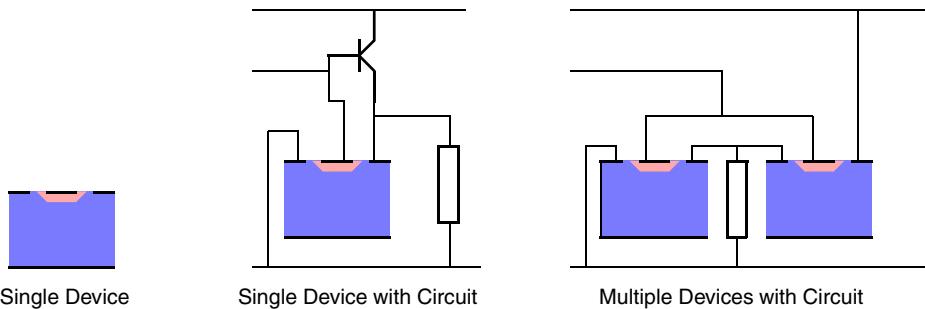


Figure 2 Three types of simulation

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 (Mesh and 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 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 \AA) 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 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, Tecplot SV is used to visualize the output from the simulation in 2D and 3D, and Inspect is used to plot the electrical characteristics.

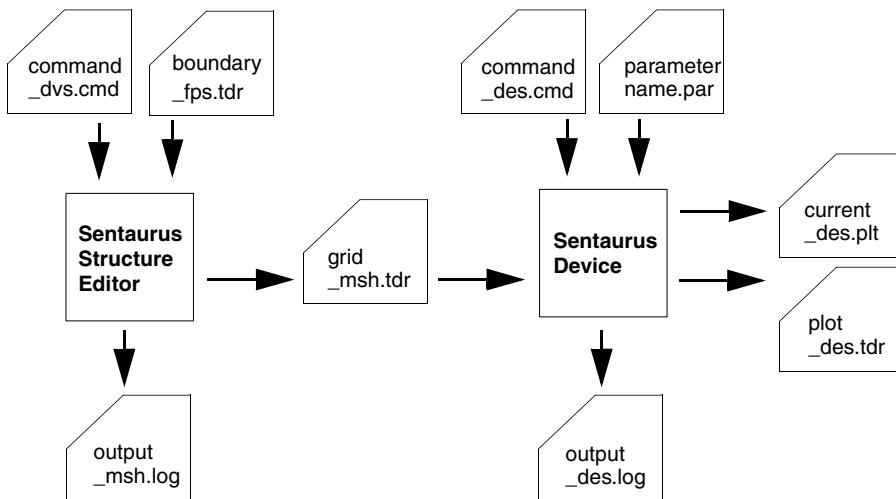


Figure 3 Typical tool flow with device simulation using Sentaurus Device

Starting Sentaurus Device

There are two ways to start Sentaurus Device: from the command line or Sentaurus Workbench.

From 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 command-line option `-ver`. For example:

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

starts version 1.4 of Sentaurus Device. To run the latest version in a particular release, use the command-line option `-rel`. For example:

```
sdevice -rel C-2009.06 nmos_des.cmd
```

starts the latest version available in release C-2009.06. To run a particular version in a particular release, combine `-ver` and `-rel`. For example:

```
sdevice -rel 10.0 -ver 10.0.5 nmos_des.cmd
```

starts Sentaurus Device, release 10.0, version 10.0.5.

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

[Appendix D on page 1053](#) lists the command options of Sentaurus Device, which include:

```
sdevice -versions
```

Checks which versions are in the installation path.

```
sdevice -P
```

Extracts model parameter files (see [Generating a Copy of Parameter File on page 115](#)).

```
sdevice -L
```

Extracts model parameter library (see [Library of Materials on page 117](#)).

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

Prints parameter names that can be ramped (see [Ramping Physical Parameter Values on page 71](#)).

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 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..., and #endif...).

To access the tutorial examples for Sentaurus Device, open Sentaurus Workbench and either select **Help > Training** or click the corresponding toolbar button. The Sentaurus Training opens in a separate window.

Simulation Examples

In the following sections, many of the widely used Sentaurus Device commands are introduced in the context of a series of typical MOSFET device simulations. The examples are available as Sentaurus Workbench projects, in the directory \$STROOT/tcad/\$STRELEASE/lib/sdevice/GettingStarted.

First, a very simple example of an I_d - V_g simulation is presented using default models and methods (`simple_Id-Vg`). Second, a more advanced approach to an I_d - V_d simulation (`advanced_Id-Vd`) is presented, in which more complex models are introduced and some important options are indicated.

Subsequently, a mixed-mode simulation of a CMOS inverter is presented (`advanced_Inverter`), and the small-signal AC response of a MOSFET is obtained (`advanced_AC`). The intention is to introduce some of the most widely used Sentaurus Device features in a realistic context.

Example: Simple MOSFET Id–Vg Simulation

Simulation of the drain current–gate voltage characteristic of a MOSFET is a typical Sentaurus Device application. It allows important device properties, such as threshold voltage, off-current, subthreshold slope, on-state drive current, and transconductance to be extracted. In this example, only the most essential commands are used for a reasonable simulation.

Input Command File

The Sentaurus Device command file is organized in command or statement sections that can be in any order (except in mixed-mode simulations). Sentaurus Device keywords are not case sensitive and most can be abbreviated. However, Sentaurus Device is syntax sensitive, for example, parentheses must be consistent and character strings for variable names must be delimited by quotation marks (" ").

An example of a complete command file (`sdevice_des.cmd` in the Sentaurus Workbench project `simple_Id-Vg`) is presented. Each statement section is explained individually.

```

File {
    * input files:
    Grid      = "nmos_msh.tdr"
    * output files:
    Plot      = "n1_des.tdr"
    Current   = "n1_des.plt"
    Output    = "n1_des.log"
}

Electrode {
    { Name="source"  Voltage=0.0 }
    { Name="drain"   Voltage=0.1 }
    { Name="gate"     Voltage=0.0 Barrier=-0.55 }
    { Name="substrate" Voltage=0.0 }
}

Physics {
    Mobility (DopingDependence HighFieldSat Enormal)
    EffectiveIntrinsicDensity (BandGapNarrowing (OldSlotboom))
}

Plot {
    eDensity hDensity eCurrent hCurrent
    Potential SpaceCharge ElectricField
    eMobility hMobility eVelocity hVelocity
    Doping DonorConcentration AcceptorConcentration
}

Math {
    Extrapolate
    RelErrControl
}

Solve {
    #-initial solution:
    Poisson
}

```

1: Overview

Example: Simple MOSFET Id–Vg Simulation

```
Coupled { Poisson Electron }
#-ramp gate:
Quasistationary ( MaxStep=0.05
    Goal{ Name="gate" Voltage=2 } )
    { Coupled { Poisson Electron } }
}

$STROOT/tcad/$STRELEASE/lib/sdevice/GettingStarted/simple_Id-Vd/
sdevice_des.cmd
```

File Section

First, the input files that define the device structure and the output files for the simulation results must be specified. The device to be simulated is the one plotted in [Figure 1 on page 4](#). The device is defined by the file `nmos_msh.tdr`:

```
File {
    * input files:
    Grid      = "nmos_msh.tdr"
    * output files:
    Plot      = "n1_des.tdr"
    Current   = "n1_des.plt"
    Output    = "n1_des.log"
}
```

The File section specifies the input and output files necessary to perform the simulation.

* input files:

This is a comment line.

`Grid = "nmos_msh.tdr"`

This essential input file (default extension `.tdr`) defines the mesh and various regions of the device structure, including contacts. Sentaurus Device automatically determines the dimensionality of the problem from this file. It also contains the doping profiles data for the device structure.

* output files:

This is a comment line.

`Plot = "n1_des.tdr"`

This is the file name for the final spatial solution variables on the structure mesh (extension `_des.tdr`).

```
Current = "n1_des.plt"
```

This is the file name for electrical output data (such as currents, voltages, charges at electrodes). Its standard extension is `_des.plt`.

```
Output = "n1_des.log"
```

This is an alternate file name for the output log or protocol file (default name `output_des.log`) that is automatically created whenever Sentaurus Device is run. This file contains the redirected standard output, which is generated by Sentaurus Device as it runs.

NOTE Only the root file names are necessary. Sentaurus Device automatically appends the appropriate file name extensions, for example, `Plot="n1"` is sufficient.

The device in this example is two-dimensional. By default, Sentaurus Device assumes a ‘thickness’ (effective gate width along the z-axis) of 1 μm . This effective width is adjusted by specifying an `AreaFactor` in the `Physics` section, or an `AreaFactor` for each electrode individually. An `AreaFactor` is a multiplier for the electrode currents and charges.

Electrode Section

Having loaded the device structure into Sentaurus Device, it is necessary to specify which of the contacts are to be treated as electrodes. Electrodes in Sentaurus Device are defined by electrical boundary conditions and contain no mesh. For example, in [Figure 7 on page 17](#), the ‘polysilicon’ gate is empty; it is not a region.

The `Electrode` section defines all the electrodes to be used in the Sentaurus Device simulation, with their respective boundary conditions and initial biases. Any contacts that are not defined as electrodes are ignored by Sentaurus Device. The polysilicon gate of a MOS transistor can be treated in two ways:

- As a metal, in which case, it is simply an electrode.
- As a region of doped polysilicon, in which case, the gate electrode must be a contact on top of the polysilicon region.

In the former case, an important property of the gate electrode is the ‘metal’–semiconductor work function difference. In Sentaurus Device, this is defined by the parameter `barrier`, which equals the difference in energy [eV] between the polysilicon extrinsic Fermi level and the intrinsic Fermi level in the silicon. The value of `barrier` must, therefore, be specified to be consistent with the doping in the polysilicon. This is the gate definition used in this example and is valid for most applications. However, it totally neglects any polysilicon depletion effects.

1: Overview

Example: Simple MOSFET Id–Vg Simulation

In the latter case, where the gate is modeled as an appropriately doped polysilicon region, the contact must be on top of the polysilicon and Ohmic (the default condition). In this case, depletion of the polysilicon is modeled correctly.

```
Electrode{  
    { Name="source" Voltage=0.0 }  
    { Name="drain" Voltage=0.1 }  
    { Name="gate"   Voltage=0.0 Barrier=-0.55 }  
    { Name="substrate" Voltage=0.0 }  
}
```

Name="string"

Each electrode is specified by a case-sensitive name that must match exactly an existing contact name in the structure file. Only those contacts that are named in the Electrode section are included in the simulation.

Voltage=0.0

This defines a voltage boundary condition with an initial value. One or more boundary conditions must be defined for each electrode, and any value given to a boundary condition applies in the initial solution. In this example, the simulation commences with a 100 mV bias on the drain.

Barrier=-0.55

This is the metal–semiconductor work function difference or barrier value for a polysilicon electrode that is treated as a metal. This is defined, in general, as the difference between the metal Fermi level in the electrode and the intrinsic Fermi level in the semiconductor. This barrier value is consistent with n⁺-polysilicon doping.

Physics Section

The Physics section allows a selection of the physical models to be applied in the device simulation. In this example, it is sufficient to include basic mobility models and a definition of the band gap (and, therefore, the intrinsic carrier concentration).

Potentially important effects, such as impact ionization (avalanche breakdown at the drain), are ignored at this stage.

```
Physics {  
    Mobility (DopingDependence HighFieldSat Enormal)  
    EffectiveIntrinsicDensity (BandGapNarrowing (OldSlotboom))  
}
```

Mobility (DopingDependence HighFieldSat Enormal)

Mobility models including doping dependence, high-field saturation (velocity saturation), and transverse field dependence are specified for this simulation.

NOTE HighFieldSaturation can be specified for a specific carrier (for example, eHighFieldSaturation for electrons) and is a function of the effective field experienced by the carrier in its direction of motion. Sentaurus Device provides a choice of effective field computation: GradQuasiFermi (the default), Eparallel, or CarrierTempDrive (in hydrodynamic simulations only).

EffectiveIntrinsicDensity (BandGapNarrowing (OldSlotboom))

This is the silicon bandgap narrowing model that determines the intrinsic carrier concentration.

Plot Section

The Plot section specifies all of the solution variables that are saved in the output plot files (.tdr). Only data that Sentaurus Device is able to compute, based on the selected physics models, is saved to a plot file.

```
Plot {
    eDensity hDensity eCurrent hCurrent
    Potential SpaceCharge ElectricField
    eMobility hMobility eVelocity hVelocity
    Doping DonorConcentration AcceptorConcentration
}
```

An extensive list of optional plot variables is in [Appendix F on page 1059](#).

To save a variable as a vector, append /Vector to the keyword:

```
Plot { eCurrent/Vector ElectricField/v }
```

Math Section

Sentaurus Device solves the device equations (which are essentially a set of partial differential equations) self-consistently, on the discrete mesh, in an iterative fashion. For each iteration, an error is calculated and Sentaurus Device attempts to converge on a solution that has an acceptably small error. For this example, it is only necessary to define a few settings for the numeric solver. Other options, including selection of solver type and user definition of error criteria, are outlined in the [Math Section on page 91](#).

1: Overview

Example: Simple MOSFET Id–Vg Simulation

```
Math {  
    Extrapolate  
    RelErrControl  
}
```

Extrapolate

In quasistationary bias ramps, the initial guess for a given step is obtained by extrapolation from the solutions of the previous two steps (if they exist).

RelErrControl

Switches error control during iterations from using internal error parameters to more physically meaningful parameters (`ErrRef`) (see [Math Section on page 91](#)).

Solve Section

The `Solve` section defines a sequence of solutions to be obtained by the solver. The drain has a fixed initial bias of 100 mV, and the source and substrate are at 0 V. To simulate the $I_d V_g$ characteristic, it is necessary to ramp the gate bias from 0 V to 2 V, and obtain solutions at a number of points in-between. By default, the size of the step between solution points is determined by Sentaurus Device internally, see [Quasistationary Command on page 67](#).

As the simulation proceeds, output data for each of the electrodes (currents, voltages, and charges) is saved to the current file `n1_des.plt` after each step and, therefore, the electrical characteristic is obtained. This can be plotted using `Inspect`, as shown in [Figure 4 on page 16](#) and [Figure 5 on page 16](#). The final 2D solution is saved in the plot file `n1_des.tdr`, which is plotted in [Figure 6 on page 17](#) and [Figure 7 on page 17](#).

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

Poisson

This specifies that the initial solution is of the nonlinear Poisson equation only. Electrodes have initial electrical bias conditions as defined in the `Electrode` section. In this example, a 100 mV bias is applied to the drain.

```
Coupled {Poisson Electron}
```

The second step introduces the continuity equation for electrons, with the initial bias conditions applied. In this case, the electron current continuity equation is solved fully coupled to the Poisson equation, taking the solution from the previous step as the initial guess. The fully coupled or ‘Newton’ method is fast and converges in most cases. It is rarely necessary to use a ‘Plugin’ (or the so-called Gummel) approach.

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

The Quasistationary statement specifies that quasi-static or steady state ‘equilibrium’ solutions are to be obtained. A set of Goals for one or more electrodes is defined in parentheses. In this case, a sequence of solutions is obtained for increasing gate bias up to and including the goal of 2 V. A fully coupled (Newton) method for the self-consistent solution of the Poisson and electron continuity equations is specified in braces. Each bias step is solved by taking the solution from the previous step as its initial guess. If `Extrapolate` is specified in the `Math` section, the initial guess for each bias step is calculated by extrapolation from the previous two solutions.

Simulated Id–Vg Characteristic

In Inspect, the gate `OuterVoltage` is plotted to the x-axis, and the drain `eCurrent` is plotted to the y-axis. The gate `InnerVoltage` is an internally calculated voltage equal to the `OuterVoltage` and adjusted to account for the barrier, and it is not plotted.

NOTE Although the solution points obtained are equally spaced (0.1 V), this is not predetermined. Generally, Sentaurus Device selects step sizes according to an internal algorithm for maximum numeric robustness. If a step fails to converge, the step size is reduced until convergence is achieved. In this example, the simulation proceeded with the maximum step size from start to finish.

1: Overview

Example: Simple MOSFET Id–Vg Simulation

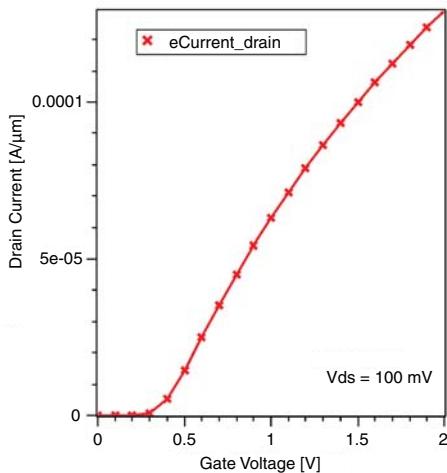


Figure 4 I_d – V_{gs} characteristic of 0.18 μm n-channel MOSFET

In [Figure 5](#), a log(I_d)-lin(V_{gs}) plot shows the subthreshold characteristic.

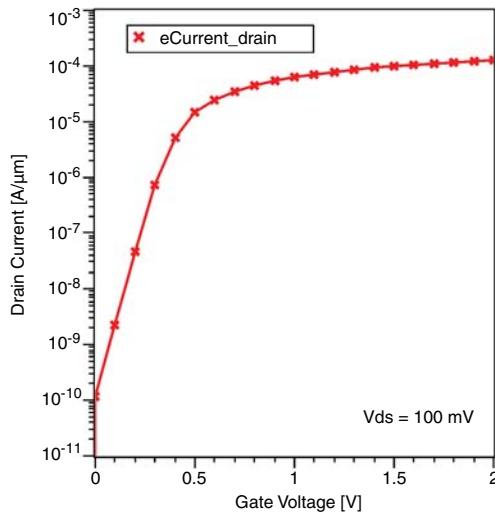


Figure 5 I_d – V_{gs} characteristic of 0.18 μm n-channel MOSFET replotted on semi-log scale

NOTE Successful completion of the simulation is confirmed by a message in the output file:

Finished, because of...

Curve trace finished.

Writing plot 'n1_des.tdr' (TDR format) ... done.

The plot data in `n1_des.tdr` can be analyzed using a graphics package such as Tecplot SV.

Analysis of 2D Output Data

The contents of the file n1_des.tdr is loaded into Tecplot SV. The command is:

```
> tecplot_sv n1_des.tdr &
```

[Figure 6](#) shows the default display, which is electrostatic potential.

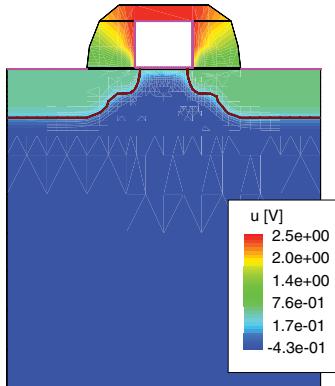


Figure 6 Default display of electrostatic potential and junctions

The electrostatic potential is relative to the intrinsic Fermi level in the silicon.

NOTE The potential at the gate electrode is 2.55 V (`InnerVoltage`), which equals the applied bias minus the barrier potential. In the substrate, which is tied to 0 V, the ‘inner’ electrostatic potential is –0.4253 V, which corresponds to the position of the hole quasi-Fermi level relative to the intrinsic level.

[Figure 7](#) is a magnification of the channel region, created by selecting the dataset `eDensity`. The inversion layer is clearly visible.

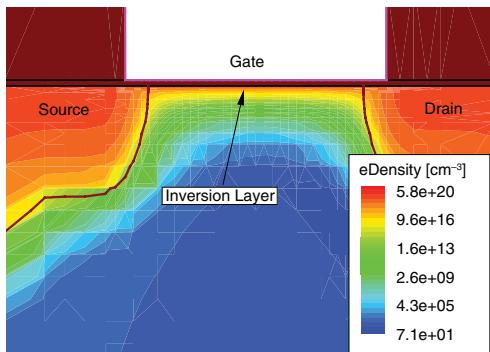


Figure 7 Magnification of channel region showing contours of electron concentration

1: Overview

Example: Advanced Hydrodynamic Id–Vd Simulation

Figure 8 shows the electron and hole densities along a vertical cutline in the center of the channel.

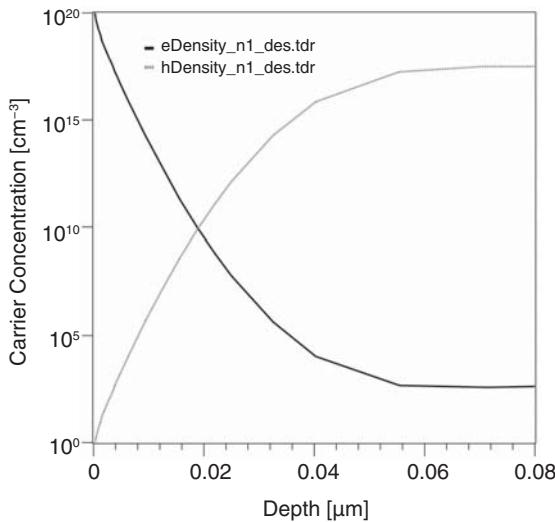


Figure 8 Vertical electron and hole profiles at center of channel

Example: Advanced Hydrodynamic Id–Vd Simulation

Input Command File

```
#-----#
#- Sentaurus Device command file for
#-
#- Id=f(Vd) for Vd=0-10V while Vg=[0.0V, 1.0V, 2.0V] and Vs=0V
#-
#- USING HYDRODYNAMIC MODEL & IMPACT IONIZATION - FAMILY OF CURVES
#-----#
File {
    Grid      = "@tdr@"
    Parameter = "mos"
    Plot      = "@tdrdat@"
    Current   = "@plot@"
    Output    = "@log@"
}

Electrode {
    { Name="source" Voltage=0.0 }
    { Name="drain"  Voltage=0.0 }
    { Name="gate"   Voltage=0.0 Barrier=-0.55 }
```

```

        { Name="substrate" Voltage=0.0 }
}

Physics {
    AreaFactor=0.4
    Hydrodynamic( eTemperature )
    Mobility ( DopingDependence Enormal
                eHighFieldsat(CarrierTempDrive)
                hHighFieldsat(GradQuasiFermi) )
    Recombination( SRH(DopingDependence)
                    eAvalanche(CarrierTempDrive)
                    hAvalanche(Eparallel) )
    EffectiveIntrinsicDensity (BandGapNarrowing (OldSlotboom))
}

Physics(
    MaterialInterface="Silicon/Oxide") {
    charge(Conc=4.5e+10)
}

Plot {
    eDensity hDensity eCurrent hCurrent
    equasiFermi hquasiFermi
    eTemperature
    ElectricField eParallel hParallel
    Potential SpaceCharge
    SRHRecombination Auger AvalancheGeneration
    eMobility hMobility eVelocity hVelocity
    Doping DonorConcentration AcceptorConcentration
}

CurrentPlot {
    Potential (82, 530, 1009)
    eTemperature (82, 530, 1009)
}

Math {
    Extrapolate
    RelErrControl
    Iterations=20
    NotDamped=50
    BreakCriteria {Current(Contact="drain" Absval=3e-4)
    }
}

Solve {
    # initial gate voltage Vgs=0.0V
    Poisson
}

```

1: Overview

Example: Advanced Hydrodynamic Id-Vd Simulation

```
Coupled { Poisson Electron }
Coupled { Poisson Electron Hole eTemperature }
Save (FilePrefix="vg0")

# ramp gate and save solutions:

# second gate voltage Vgs=1.0V
Quasistationary
(InitialStep=0.1 Maxstep=0.1 MinStep=0.01
Goal { name="gate" voltage=1.0 } )
{ Coupled { Poisson Electron Hole eTemperature } }
Save(FilePrefix="vg1")

# third gate voltage Vgs=2.0V
Quasistationary
(InitialStep=0.1 Maxstep=0.1 MinStep=0.01
Goal { name="gate" voltage=2.0 } )
{ Coupled { Poisson Electron Hole eTemperature } }
Save(FilePrefix="vg2")

# Load saved structures and ramp drain to create family of curves:

# first curve
Load(FilePrefix="vg0")
NewCurrentPrefix="vg0_"
Quasistationary
(InitialStep=0.01 Maxstep=0.1 MinStep=0.0001
Goal{ name="drain" voltage=10.0 }
)
{ Coupled {Poisson Electron Hole eTemperature}
CurrentPlot (time=
(range = (0 0.2) intervals=20;
range = (0.2 1.0)))}

# second curve
Load(FilePrefix="vg1")
NewCurrentPrefix="vg1_"
Quasistationary
(InitialStep=0.01 Maxstep=0.1 MinStep=0.0001
Goal{ name="drain" voltage=10.0 }
)
{Coupled {Poisson Electron Hole eTemperature}
CurrentPlot (time=
(range = (0 0.2) intervals=20;
range = (0.2 1.0)))}
```

```

# third curve
Load(FilePrefix="vg2")
NewCurrentPrefix="vg2_"
Quasistationary
(InitialStep=0.01 Maxstep=0.1 MinStep=0.0001
Goal{ name="drain" voltage=10.0 }
)
{ Coupled {Poisson Electron Hole eTemperature }
CurrentPlot (time=
(range = (0 0.2) intervals=20;
range = (0.2 1.0)))
}
$STROOT/tcad/$STRELEASE/lib/sdevice/GettingStarted/advanced_Id-Vd/
sdevice_des.cmd

```

File Section

```

File {
    Grid      = "@tdr@"
    Parameter = "mos"
    Plot      = "@tdrdat@"
    Current   = "@plot@"
    Output    = "@log@"
}

```

The File section specifies the input and output files necessary to perform the simulation. In the example, all file names except mos are file references, which Sentaurus Workbench recognizes and replaces with real file names during preprocessing. For example, in the processed command file pp2_des.cmd:

```

File {
    Grid      = "n1_msh.tdr"
    Parameter = "mos"
    Plot      = "n2_des.tdr"
    Current   = "n2_des.plt"
    Output    = "n2_des.log"
}

```

Grid

The given file names relate to the appropriate node numbers in the Sentaurus Workbench Family Tree. In most projects, Grid is generated by Mesh and, therefore, has the characteristic name nX_msh.tdr.

1: Overview

Example: Advanced Hydrodynamic Id–Vd Simulation

Plot, Current, Output

See [File Section on page 10](#).

Parameter = "mos"

The optional input file mos.par contains user-defined values for model parameters (coefficients) (see [Parameter File](#)). Sentaurus Device adds the file extension .par automatically.

Main Options

(e|h)Lifetime = "<name>"

Loads a lifetime profile contained in a data file. The profile is generated using Sentaurus Structure Editor or Mesh.

Parameter File

The parameter file contains user-defined values for model parameters (coefficients). The parameters in this file replace the values contained in a default parameter file models.par. All other parameter values remain equal to the defaults in models.par. Model coefficients can be specified separately for each region or material in the device structure (see [Specifying Region and Material Parameters on page 115](#)). Although this feature is intended for the simulation of heterostructure devices, it is useful in silicon devices as it allows materials, such as polysilicon, to have different mobilities.

The default parameter file contains the default parameters for all the physical models available in Sentaurus Device. A copy of the parameter file for silicon is extracted using the command:

```
sdevice -P
```

A list of the parameter files is printed to the command window and into a file models.par, which is created in the working directory. For other materials, such as gallium arsenide (GaAs) and silicon carbide (SiC), use:

```
sdevice -P:GaAs  
sdevice -P:SiC
```

More options are described in [Generating a Copy of Parameter File on page 115](#).

Listing of mos.par

```
Scharfetter * SRH recombination lifetimes
{ * tau=taumin+(taumax-taumin) / ( 1+(N/Nref )^gamma)
  *           electrons      holes
  taumin = 0.0000e+00, 0.0000e+00 # [s]
  taumax = 1.0000e-07, 1.0000e-07 # [s]
}
```

Report in Protocol File n3_des.log

```
Reading parameter file 'mos.par' ...
Differences compared with default parameters:
Scharfetter(elec): tau_max = 1.0000e-07, instead of: 1.0000e-05 [s]
Scharfetter(hole): tau_max = 1.0000e-07, instead of: 3.0000e-06 [s]
```

Electrode Section

```
Electrode{
  { Name="source"  Voltage=0.0 }
  { Name="drain"   Voltage=0.0 }
  { Name="gate"     Voltage=0.0 Barrier=-0.55 }
  { Name="substrate" Voltage=0.0 }
}
```

In this example, all electrodes have voltage boundary conditions with the initial condition of zero bias.

Main Options

Current=

Defines a current boundary condition with initial value [A].

Charge=

Defines a floating electrode with a charge boundary condition and an initial charge value [C].

Resist=

Defines a series resistance [Ω] (AreaFactor-dependent).

1: Overview

Example: Advanced Hydrodynamic Id–Vd Simulation

eRecVelocity=

Defines a recombination velocity [cm/s] at a contact for electrons (hRecVelocity for holes).

Schottky=

Defines an electrode as a Schottky contact. The attributes of such a contact are specified as Barrier and eRecVelocity or hRecVelocity.

AreaFactor=

Specifies a multiplication factor for the current in or out of an electrode. It is preferable to define AreaFactor in the Physics section. In a 2D simulation, this can represent the size of the device in the third dimension (for example, gate width), which is 1 μm by default.

Barrier=-0.55

This is the metal–semiconductor work function difference or barrier value for an electrode that is treated as a metal. Defined, in general, as the difference between the metal Fermi level in the electrode and the intrinsic Fermi level in the semiconductor.

In this example, this corresponds to the difference between the quasi-Fermi level in the gate polysilicon and the intrinsic Fermi level in the silicon. The value of barrier=-0.55 is approximately representative of n⁺-doped polysilicon.

Physics Section

```
Physics {
    AreaFactor=0.4
    Hydrodynamic(eTemperature)
    Mobility(DopingDependence Enormal
        hHighFieldSaturation(GradQuasiFermi)
        eHighFieldSaturation(CarrierTempDrive))
    Recombination(SRH(DopingDependence)
        eAvalanche(CarrierTempDrive)
        hAvalanche(Eparallel))
    EffectiveIntrinsicDensity(BandGapNarrowing (OldSlotboom))
}
```

AreaFactor=0.4

Specifies that the electrode currents and charges are multiplied by a factor of 0.4 (equivalent to a simulated gate width of 0.4 μm).

Hydrodynamic (eTemperature)

Selects hydrodynamic transport models for electrons only. Hole transport is modeled using drift-diffusion.

Mobility (DopingDependence Enormal

See [Physics Section on page 12](#).

hHighFieldSaturation (GradQuasiFermi)

Velocity saturation for holes. It uses the default model after Canali (based on Caughey–Thomas) (see [High-Field Saturation on page 294](#)), and is driven by a field computed as the gradient of the hole quasi-Fermi level, which is the default.

eHighFieldSaturation (CarrierTempDrive)

Velocity saturation for electrons. It is based on an adaptation of the Canali model, driven by an effective field that is based on the electron temperature (kinetic energy) (see [High-Field Saturation on page 294](#)).

Recombination (. . .)

Defines the generation and recombination models.

SRH (DopingDependence)

Shockley–Read–Hall recombination with doping-dependent lifetime (Scharfetter coefficients modified in the parameter file mos.par).

eAvalanche (CarrierTempDrive)

Avalanche multiplication for electrons is driven by an effective field computed from the local carrier temperature.

hAvalanche (Eparallel)

Avalanche multiplication for holes is driven by the component of the field that is parallel to the hole current flow. The default impact ionization model is from van Overstraeten–de Man (see [Avalanche Generation on page 324](#)).

Main Options

Temperature

Specifies the lattice temperature [K] (default 300 K).

1: Overview

Example: Advanced Hydrodynamic Id–Vd Simulation

IncompleteIonization

Incomplete ionization of individual species.

GateCurrent (<model>)

Selects a model for gate leakage or (dis)charging of floating gates (see [Diffusion Noise on page 501](#)).

Recombination (Band2Band)

Simulates band-to-band tunneling.

NOTE Model coefficients can be specified independently for each region or material in the device structure. You can specify the model coefficients in the parameter file .par.

Interface Physics

```
Physics(MaterialInterface="Silicon/Oxide") {  
    Charge(Conc=4.5e+10)  
}
```

Special physical models are defined for the interfaces between specified regions or materials. In this example, an interface fixed charge is specified for all oxide–silicon interfaces with areal concentration defined in cm⁻².

Main Options

Interface traps can be specified and interfaces can be defined between materials or specific regions:

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

Plot Section

```
Plot {
    eDensity hDensity
    eCurrent hCurrent
    eQuasiFermi hQuasiFermi
    eTemperature
    ElectricField eEparallel hEparallel
    Potential SpaceCharge
    SRHRecombination Auger AvalancheGeneration
    eMobility hMobility eVelocity hVelocity
    Doping DonorConcentration AcceptorConcentration
}
```

The variables `eTemperature` and `hEparallel` are added to the list of variables to be included in the plot file `_des.tdr`. The data is saved only if the variables are consistent with the specified physical models.

CurrentPlot Section

```
CurrentPlot {
    Potential (82, 530, 1009)
    eTemperature (82, 530, 1009)
}
```

This feature allows solution variables at specified nodes to be saved to the current file `_des.plt`.

In this example, the electrostatic potential and electron temperature are saved at three nodes corresponding to selected locations in the source (# 82), drain extensions (# 530), and the center of the body (# 1009). Node numbers are identified using the `VertexIndex` variable in Tecplot SV (see [Tecplot SV User Guide, Environment Variables on page 48](#)). Any number of nodes is allowed.

Main Options

Any of the variables in [Table 121 on page 1060](#).

1: Overview

Example: Advanced Hydrodynamic Id–Vd Simulation

Math Section

```
Math {  
    Extrapolate  
    RelErrControl  
    NotDamped=50  
    Iterations=20  
    BreakCriteria {Current (Contact="drain" Absval=3e-4)}  
}
```

NotDamped=50

Specifies the number of Newton iterations over which the right-hand side (RHS)-norm is allowed to increase. With the default of 1, the error is allowed to increase for one step only. It is recommended that `NotDamped > Iterations` is set to allow a simulation to continue despite the RHS-norm increasing.

Iterations=20

Specifies the maximum number of Newton iterations allowed per bias step (default=50). If convergence is not achieved within this number of steps, for a quasistationary or transient simulation, the step size is reduced by the factor `decrement` (see [Quasistationary Command on page 67](#)) and simulation continues.

BreakCriteria {Current (Contact="drain" Absval=3e-4)}

Break criteria are used to stop a simulation if a certain limit value is exceeded (see [Break Criteria on page 99](#)). In this case, the simulation terminates when the drain current exceeds 3×10^{-4} A.

Example

Cylindrical (<float>)

This keyword forces a 2D device to be simulated using cylindrical coordinates, that is, it is rotated around the y-axis. The optional argument <float> is the x-location of the axis of symmetry (default=0).

Method=

Selects the linear solver to be used in the coupled command.

Break criteria based on bulk properties (not contact variables) can be defined in a material-specific or region-specific Math section:

```
Math (material="Silicon") {
    BreakCriteria { LatticeTemperature (Maxval = 1400)
        CurrentDensity (Absval = 1e7) }
}
```

Solve Section

```
Solve {
    # initial gate voltage Vgs=0.0V
    Poisson
    Coupled { Poisson Electron }
    Coupled { Poisson Electron Hole eTemperature }
    Save(FilePrefix="vg0")
```

The Solve section defines the sequence of solutions to be obtained by the solver.

Poisson

Specifies the initial solution of the nonlinear Poisson equation. Electrodes will have initial electrical bias conditions as defined in the Electrode section.

In this example, all electrodes are at zero initial bias. However, the initial conditions can be nonzero. For example, it is reasonable to begin with a small bias applied to the gate or drain of a MOSFET.

Coupled { Poisson Electron }

The second step introduces carrier continuity for electrons, with the initial bias conditions still applied. In this case, the electron current continuity is solved fully coupled to the Poisson equation.

Coupled { Poisson Electron Hole eTemperature }

Solves the carrier continuity equations for both carriers and the electron temperature equations.

Save (FilePrefix="vg0")

The zero bias solution is saved to a file named with the default extension `_des.sav`, in this case, `vg0_des.sav`.

1: Overview

Example: Advanced Hydrodynamic Id–Vd Simulation

The save file contains all the information required to restart the simulation, the solution variables on the mesh, and the bias conditions on the electrodes. It can be reloaded within the same Solve section or in another simulation file. In the latter case, the model selection must be consistent.

Solve Continued

```
# ramp gate and save solutions:  
  
# second gate voltage Vgs=1.0V  
Quasistationary  
( Goal { Name="gate" Voltage=1.0 }  
InitialStep=0.1 Maxstep=0.1 MinStep=0.01  
)  
{ Coupled { Poisson Electron Hole eTemperature } }  
Save(FilePrefix="vg1"){
```

The Quasistationary statement implies that a series of quasistatic or steady state ‘equilibrium’ solutions can be obtained.

```
( Goal { Name="gate" Voltage=1.0 } )
```

A Goal or set of Goals for one or more electrodes are defined in the parentheses. In this case, the gate bias is increased to and includes the goal of 1 V.

```
(InitialStep=0.1 Maxstep=0.1 MinStep=0.01
```

Specifies the constraints on the step size (Δt) as proportions of the normalized Goal ($t=1$). With the initial and maximum step sizes set to 0.1 ($t=0.1$), the gate voltage ramp is concluded in a total of ten steps, not counting the ‘zero’ step. It is assumed that convergence is achieved at each step.

If, at any step, there is a failure to converge, Sentaurus Device performs automatic step size reduction until convergence is again achieved, and then continues the simulation.

```
{ Coupled { Poisson Electron Hole eTemperature } }
```

At each step, the device equations are solved self-consistently (coupled or Newton method). Poisson, hole current continuity, electron flux and temperature continuity are specified in braces.

```
Save(FilePrefix="vg1") {
```

At the end of the ramp, another save file is created for the 1 V gate bias solution, vg1_des.sav. Next, the gate bias is ramped to 2 V to provide a solution file, vg2_des.sav.

Solve Continued

```
# Load solutions & ramp drain for family of curves
Load(FilePrefix="vg0")
NewCurrentPrefix="vg0_"
Quasistationary
  (Goal { Name="drain" Voltage=10.0 }
   InitialStep=0.01 Maxstep=0.1 MinStep=0.0001
  )
  { Coupled { Poisson Electron Hole eTemperature }
    CurrentPlot (Time =
      ( range = (0 0.2) intervals = 20;
      range = (0.2 1.0)))
  }
}
```

Load(FilePrefix="vg0")

Loads the solution file for zero gate bias.

NewCurrentPrefix="vg0_"

Starts a new current file in which the following results are saved. The file is vg0_n3_des.plt.

Quasistationary

Implies that a series of quasi-static or steady state ‘equilibrium’ solutions are to be obtained.

(Goal { Name="drain" Voltage=10.0 }

In this case, the goal is to increase the drain bias up to and including the goal of 10 V. The goal is not reached if any of the break criteria are met.

InitialStep=0.01 MaxStep=0.1 MinStep=0.0001)

Specifies the constraints on the step size (Δt) as proportions of the normalized Goal ($t=1$). If, at any step, there is a failure to converge, Sentaurus Device performs automatic step size reduction until convergence is again achieved, and then continues the simulation.

{ Coupled { Poisson Electron Hole eTemperature }

At each step, the device equations are solved self-consistently (coupled or Newton method). Poisson, hole current continuity, electron flux, and temperature continuity are specified in braces.

1: Overview

Example: Advanced Hydrodynamic Id–Vd Simulation

```
CurrentPlot (Time=(range=(0 0.2) intervals=20;
range=(0.2 1.0)))
```

This statement ensures that solutions are saved in the current file only at certain specific values of the drain voltage in the range 0 V to 2.0 V. In this example, time implies the notional (normalized) time t whose full range is $t=0$ to $t=1$.

In this case, 21 equally spaced solutions are saved in the range $t=0$ to $t=0.2$, corresponding to 0 V and 2.0 V (that is, in steps of 0.1 V). This is in addition to any intermediate steps that may be solved but not saved. From $t=0.2$ to $t=1.0$ (2.0 V to 10.0 V), all solutions are saved in the file `vg0_n3_des.plt`.

Figure 9 shows the family of drain output characteristics (I_d – V_{ds}) in the drain bias range from 0 V to 2 V. The equal drain voltage steps of 0.1 V are suitable for SPICE parameter extraction.

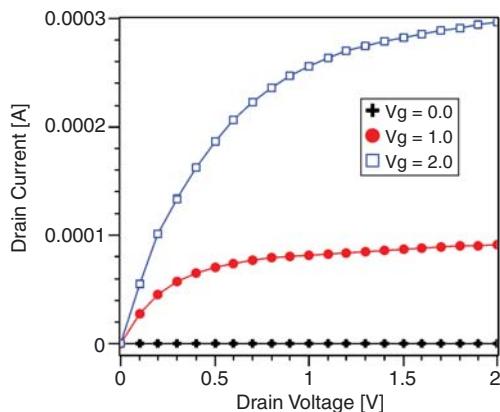


Figure 9 Family of drain output characteristics (I_d – V_{ds}) in drain bias range 0–2 V

Figure 10 shows the I_{ds} – V_{ds} characteristics plotted with the electron temperature at the drain end of the channel over the full range of the simulations.

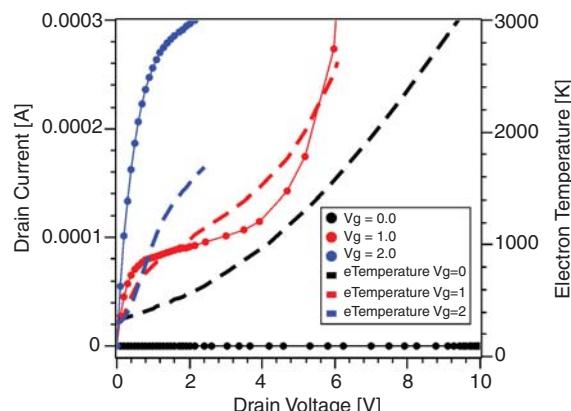


Figure 10 I_{ds} – V_{ds} characteristics plotted with electron temperature at drain end of channel over full range of simulations

Two-dimensional Output Data

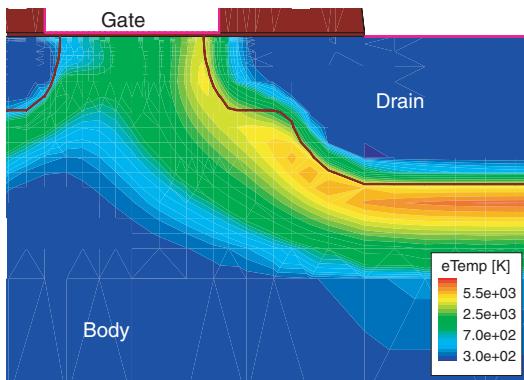


Figure 11 Contours of electron temperature computed at final solution
($V_{ds} = 2.425$ V, $I_d = 30$ mA)

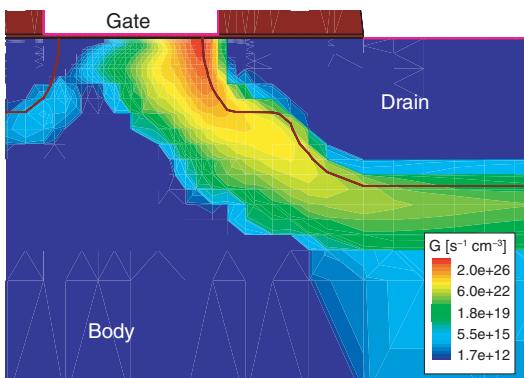


Figure 12 Contours of impact ionization rate at final solution

Example: Mixed-Mode CMOS Inverter Simulation

The mixed-mode capability of Sentaurus Device allows for the simulation of a circuit that combines any number of Sentaurus Device devices of arbitrary dimensionality (1D, 2D, or 3D) with other devices based on compact models (SPICE).

In this Sentaurus Workbench project (see `$STROOT/tcad/$STRELEASE/lib/sdevice/GettingStarted/advanced_Inverter`), a transient mixed-mode simulation is presented with two 2D physical devices; an n-channel and a p-channel MOSFET, combined with a capacitor and a voltage source to form a CMOS inverter circuit. Mesh is used to create the 2D NMOSFET and PMOSFET devices. Sentaurus Device computes the transient response of the inverter to a voltage signal, which codes a 010 binary sequence.

1: Overview

Example: Mixed-Mode CMOS Inverter Simulation

Input Command File

```
#-----#
#- Sentaurus Device command file for a transient mixed-mode simulation of the
#- switching of an inverter build with a nMOSFET and a pMOSFET.
#-----#

Device NMOS {

    Electrode {
        { Name="source"      Voltage=0.0 Area=5 }
        { Name="drain"       Voltage=0.0 Area=5 }
        { Name="gate"        Voltage=0.0 Area=5 Barrier=-0.55 }
        { Name="substrate"   Voltage=0.0 Area=5 }
    }
    File {
        Grid     = "@tdr@"
        Plot     = "nmos"
        Current  = "nmos"
        Param    = "mos"
    }
    Physics {
        Mobility( DopingDependence HighFieldSaturation Enormal )
        EffectiveIntrinsicDensity(BandGapNarrowing (OldSlotboom) )
    }
}

Device PMOS{

    Electrode {
        { Name="source"      Voltage=0.0 Area=10 }
        { Name="drain"       Voltage=0.0 Area=10 }
        { Name="gate"        Voltage=0.0 Area=10 Barrier=0.55 }
        { Name="substrate"   Voltage=0.0 Area=10 }
    }
    File {Grid = "@tdr:+1@"
        Plot     = "pmos"
        Current  = "pmos"
        Param    = "mos"
    }
    Physics {
        Mobility( DopingDependence HighFieldSaturation Enormal )
        EffectiveIntrinsicDensity(BandGapNarrowing (OldSlotboom) )
    }
}
System {
    Vsource_pset v0 (n1 n0) { pwl = (0.0e+00 0.0
                                    1.0e-11 0.0
```

```

        1.5e-11 2.0
        10.0e-11 2.0
        10.5e-11 0.0
        20.0e-11 0.0) }

NMOS nmos( "source"=n0 "drain"=n3 "gate"=n1 "substrate"=n0 )
PMOS pmos( "source"=n2 "drain"=n3 "gate"=n1 "substrate"=n2 )
Capacitor_pset c1 ( n3 n0 ){ capacitance = 3e-14 }

Set (n0 = 0)
Set (n2 = 0)
Set (n3 = 0)
Plot "nodes.plt" (time() n0 n1 n2 n3 )
}

File {
    Current= "inv"
    Output = "inv"
}
Plot {
    eDensity hDensity eCurrent hCurrent
    ElectricField eEnormal hEnormal
    eQuasiFermi hQuasiFermi
    Potential Doping SpaceCharge
    DonorConcentration AcceptorConcentration
}
Math {
    Extrapolate
    RelErrControl
    Digits=4
    Notdamped=50
    Iterations=12
    NoCheckTransientError
}
Solve {
    #-build up initial solution
    NewCurrentPrefix = "ignore_"
    Coupled { Poisson }
    Quasistationary ( InitialStep=0.1 MaxStep=0.1
        Goal { Node="n2" Voltage=2 }
        Goal { Node="n3" Voltage=2 }
    )
    { Coupled { Poisson Electron Hole } }
    NewCurrentPrefix = ""
    Unset (n3)
    Transient (
        InitialTime=0 FinalTime=20e-11
        InitialStep=1e-12 MaxStep=1e-11 MinStep=1e-15
        Increment=1.3
    )
    { Coupled { nmos.poisson nmos.electron nmos.contact

```

1: Overview

Example: Mixed-Mode CMOS Inverter Simulation

```
        pmos.poisson pmos.hole pmos.contact circuit }
    }
}
$STROOT/tcad/$STRELEASE/lib/sdevice/GettingStarted/advanced_Inverter/
sdevice_des.cmd
```

Device Section

The sequence of command sections is different when comparing mixed-mode to single-device simulation. For mixed-mode simulations, the physical devices are defined in separate `Device` statement sections. The following is the section for an n-channel MOSFET:

```
Device NMOS {
    Electrode {
        { Name="source" Voltage=0.0 Area=5 }
        { Name="drain" Voltage=0.0 Area=5 }
        { Name="gate" Voltage=0.0 Area=5 Barrier=-0.55 }
        { Name="substrate" Voltage=0.0 Area=5 }
    }
    File {
        Grid      = "@tdr@"
        Plot      = "nmos"
        Current   = "nmos"
        Param     = "mos"
    }
    Physics {
        Mobility( DopingDependence HighFieldSaturation Enormal)
        EffectiveIntrinsicDensity(BandGapNarrowing OldSlotboom )
    }
}
```

For a mixed-mode simulation (see [Device Section on page 148](#)), the physical devices are named `NMOS` and `PMOS`, and are defined in separate `Device` statements.

Inside the `Device` statements, the `Electrode`, `Physics`, and most of the `File` sections are defined in the same way as in command files for single device simulations.

NOTE The p-channel has twice the `AreaFactor` of the n-channel MOSFET, which is equivalent to setting twice the gate width.

Different physical models can be applied in each device type, as well as different coefficients if each device has a dedicated parameter file.

The equivalent section for the p-channel device is defined almost identically except that the source file for the p-channel structure is `@tdr:+1@` (referring to the Sentaurus Workbench

node corresponding to the Mesh split for the p-channel structure), and the plot and current files have the prefix `pmos`. Further, `Barrier=+0.55` for the p-channel MOSFET.

System Section

The circuit is defined in the `System` section, which uses a SPICE syntax. The two MOSFETs are connected to form a CMOS inverter with a capacitive load and voltage source for the input signal.

```

System {
    Vsource_pset v0 (n1 n0) {pwl = (0.0e+00 0.0
                                    1.0e-11 0.0
                                    1.5e-11 2.0
                                    10.0e-11 2.0
                                    10.5e-11 0.0
                                    20.0e-11 0.0)}
    NMOS nmos( "source"=n0 "drain"=n3 "gate"=n1 "substrate"=n0 )
    PMOS pmos( "source"=n2 "drain"=n3 "gate"=n1 "substrate"=n2 )

    Capacitor_pset c1 ( n3 n0 ){ capacitance = 3e-14 }
    Set (n0 = 0)
    Set (n2 = 0)
    Set (n3 = 0)
    Plot "nodes.plt" (time() n0 n1 n2 n3 )
}

Vsource_pset v0 (n1 n0) ...

```

A voltage source that generates a piecewise linear (`pwl`) voltage signal is connected between the input node (`n1`) and ground node (`n0`). The time–voltage sequence generates a low-high-low or 010 binary sequence over a 200ps time period.

`NMOS nmos (...)`

The previously defined device named `NMOS` is instantiated with a tag `nmos`. Each of its electrodes is connected to a circuit node. (If an electrode is not connected to the circuit, it is driven by any bias conditions specified in the corresponding `Electrode` statement.)

NOTE A physical device can be instantiated any number of times in a mixed-mode circuit. Therefore, it is necessary to assign a name for each instance in the circuit. In this example, the name `nmos` is chosen.

`PMOS pmos (...)`

The PMOSFET is instantiated similarly.

1: Overview

Example: Mixed-Mode CMOS Inverter Simulation

```
Capacitor_pset c1 ( n3 n0 )...
```

Capacitive load (c1) is connected between the output node (n3) and ground node (n0).

```
Set (n0 = 0)
Set (n2 = 0)
Set (n3 = 0)
```

The Set command defines the nodal voltages at the beginning of the simulation. These definitions are kept until an Unset command is specified. Node n0 is tied to the ground, and n2 and n3 are initially set to 0 V.

File Section

```
File {
    Current = "inv"
    Output = "inv"
}
```

Output file names, which are not device-specific, are defined outside of the Device sections.

Plot Section

```
Plot {
    eDensity hDensity eCurrent hCurrent
    ElectricField eEnormal hEnormal
    eQuasiFermi hQuasiFermi
    Potential Doping SpaceCharge
    DonorConcentration AcceptorConcentration
}
```

In this case, the Plot statement is global and applies to all physical devices. It can also be specified inside individual Device sections.

Math Section

```
Math {
    ...
    NoCheckTransientError
}
```

NoCheckTransientError

This keyword disables the computation of error estimates based on time derivatives. This error estimation scheme is inappropriate when abrupt changes in time are enforced externally.

In this example, it is advantageous to specify this option because the inverter is driven by a voltage pulse with very steep rising and falling edges.

Solve Section

The circuit and physical device equations are solved self-consistently for the duration of the input pulse. A transient simulation is performed for the time duration specified in the Transient command.

```
Solve {
    #-build up initial solution
    NewCurrentPrefix = "ignore_"
    Coupled { Poisson }
    Quasistationary ( InitialStep=0.1 MaxStep=0.1
        Goal { Node="n2" Voltage=2 }
        Goal { Node="n3" Voltage=2 }
    )
    { Coupled { Poisson Electron Hole } }
    NewCurrentPrefix = ""
    Unset (n3)
    Transient (
        InitialTime=0 FinalTime=20e-11
        InitialStep=1e-12 MaxStep=1e-11 MinStep=1e-15
        Increment=1.3
    )
    { Coupled { nmos.poisson nmos.electron nmos.contact
        pmos.poisson pmos.hole pmos.contact circuit } }
}
```

The initial solution is obtained in steps. It starts with the Poisson equation. Then, the electron and hole carrier continuity equations are introduced, and the nodes n2 and n3 are ramped to

1: Overview

Example: Mixed-Mode CMOS Inverter Simulation

the supply voltage of 2 V. The circuit and contact equations are included automatically with the continuity equations.

Unset (n3)

When the initial solution is established, the output node (n3) is released.

Transient (...)

In the Transient command, the start time, final time, and step size constraints (initial, maximum, minimum) are in seconds. Actual step sizes are determined internally, based on the rate of convergence of the solution at the previous step. Increment=1.3 determines the maximum step size increase.

```
{Coupled { nmos.poisson nmos.electron nmos.contact  
          pmos.poisson pmos.hole      pmos.contact  
          circuit } }
```

The Coupled statement illustrates how, in a mixed-mode environment, a specific set of equations can be selected. In this example, the electron continuity equation is solved in the NMOSFET. The hole continuity equation is solved for the PMOSFET. The corresponding contact equations for nmos and pmos and the circuit equations are added explicitly.

Results of Inverter Transient Simulation

The simulation results are plotted automatically using Inspect, driven by the command file pp3_ins.cmd, which is created (or preprocessed) by Sentaurus Workbench from the root command file inspect_ins.cmd. In [Figure 13](#), the transient response of the output voltage and drain current through the n-channel device is plotted, overlaying the input pulse.

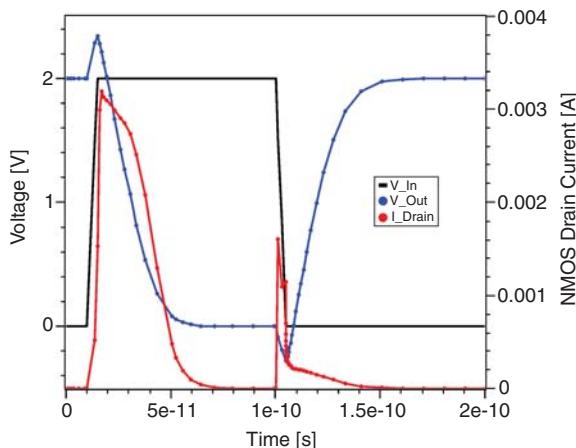


Figure 13 Inspect output from Sentaurus Workbench project (GettingStarted/advanced_Inverter)

Example: Small-Signal AC Extraction

This example demonstrates how to perform an AC analysis simulation for an NMOSFET. In Sentaurus Device, AC simulations are performed in mixed mode. In an AC simulation, Sentaurus Device computes the complex (small signal) admittance Y matrix. This matrix specifies the current response at a given node to a small voltage signal at another node:

$$j = Yu = Au + i\omega Cu \quad (1)$$

where j is the vector containing the small-signal currents at all nodes and u is the corresponding voltage vector.

Sentaurus Device output contains the components of the conductance matrix A and the capacitance matrix C (see [ACCoupled: Small-Signal AC Analysis on page 161](#)). The conductances and capacitances are used to construct the small signal equivalent circuit or to compute the other AC parameters, such as H , Z , and S .

Input Command File

```
#-----#
#- Sentaurus Device command file for
#- AC analysis at 1 MHz while Vg=-2 to 3V and Vd=2V
#-----#
Device NMOS {
    Electrode {
        { Name="source"      Voltage=0.0 }
        { Name="drain"       Voltage=0.0 }
        { Name="gate"        Voltage=0.0 Barrier=-0.55 }
        { Name="substrate"   Voltage=0.0 }
    }
    File {
        Grid      = "@tdr@"
        Current   = "@plot@"
        Plot      = "@tdrdat@"
        Param     = "mos"
    }
    Physics {
        Mobility ( DopingDependence HighFieldSaturation Enormal )
        EffectiveIntrinsicDensity(BandGapNarrowing (OldSlotboom) )
    }
    Plot {
        eDensity hDensity eCurrent hCurrent
        ElectricField eEparallel hEparallel
        eQuasiFermi hQuasiFermi
        Potential Doping SpaceCharge
    }
}
```

1: Overview

Example: Small-Signal AC Extraction

```
        DonorConcentration AcceptorConcentration
    }
}
Math {
    Extrapolate
    RelErrControl
    Notdamped=50
    Iterations=20
}
File {
    Output      = "@log@"
    ACEExtract = "@acplot@"
}
System {
    NMOS trans (drain=d source=s gate=g substrate=b)
    Vsource_pset vd (d 0) {dc=0}
    Vsource_pset vs (s 0) {dc=0}
    Vsource_pset vg (g 0) {dc=0}
    Vsource_pset vb (b 0) {dc=0}
}
Solve (
    #-a) zero solution
    Poisson
    Coupled { Poisson Electron Hole }

    #-b) ramp drain to positive starting voltage
    Quasistationary (
        InitialStep=0.1 MaxStep=0.5 MinStep=1.e-5
        Goal { Parameter=vd.dc Voltage=2 }
    )
    { Coupled { Poisson Electron Hole } }

    #-c) ramp gate to negative starting voltage
    Quasistationary (
        InitialStep=0.1 MaxStep=0.5 MinStep=1.e-5
        Goal { Parameter=vg.dc Voltage=-2 }
    )
    { Coupled { Poisson Electron Hole } }

    #-d) ramp gate -2V to +3V
    Quasistationary (
        InitialStep=0.01 MaxStep=0.04 MinStep=1.e-5
        Goal { Parameter=vg.dc Voltage=3 }
    )
    { ACCoupled (
        StartFrequency=1e6 EndFrequency=1e6
        NumberOfPoints=1 Decade
        Node(d s g b) Exclude(vd vs vg vb)
```

```
)  
  { Poisson Electron Hole }  
}  
}  
  
$STROOT/tcad/$STRELEASE/lib/sdevice/GettingStarted/advanced_AC/sdevice_des.cmd
```

Device Section

```
Device NMOS {  
  
  Electrode {  
    { Name="source" Voltage=0.0 }  
    { Name="drain" Voltage=0.0 }  
    { Name="gate" Voltage=0.0 Barrier=-0.55 }  
    { Name="substrate" Voltage=0.0 }  
  }  
  File {  
    Grid      = "@tdr@"  
    Current   = "@plot@"  
    Plot      = "@tdrdat@"  
    Param     = "mos"  
  }  
  Physics {  
    Mobility (DopingDependence HighFieldSaturation Enormal)  
    EffectiveIntrinsicDensity (BandGapNarrowing (OldSlotboom))  
  }  
  Plot {  
    eDensity hDensity eCurrent hCurrent  
    ElectricField eEparallel hEparallel  
    eQuasiFermi hQuasiFermi  
    Potential Doping SpaceCharge  
    DonorConcentration AcceptorConcentration  
  }  
}
```

The AC analysis is performed in a mixed-mode environment (see [ACCoupled: Small-Signal AC Analysis on page 161](#)). In this environment, the physical device named NMOS is defined using the Device statement.

The File section inside Device includes all device-specific files, but cannot contain the Output identifier. This identifier is outside the Device statement in a separate global File section, with the file identifier for the data file, which contains the extracted AC results (see [File Section on page 44](#)).

1: Overview

Example: Small-Signal AC Extraction

File Section

```
File {
    Output = "@log@"
    ACEExtract = "@acplot@"
}
```

Output files that are not device-specific are specified outside of the `Device` section(s).

The computed small-signal AC components are saved into a file defined by `@acplot@`, which, for a Sentaurus Device node number X, is replaced by the Sentaurus Workbench preprocessor with file name `nX_ac_des.plt`.

System Section

```
System {
    NMOS trans (drain=d source=s gate=g substrate=b)
    Vsource_pset vd (d 0) {dc=0}
    Vsource_pset vs (s 0) {dc=0}
    Vsource_pset vg (g 0) {dc=0}
    Vsource_pset vb (b 0) {dc=0}
}
```

A simple circuit is defined as a SPICE netlist in the `System` section. For a standard AC analysis, a voltage source is attached to each contact of the physical device. Each voltage source is given a different instance name.

Solve Section

```
Solve {
    #-a) zero solution
    Poisson
    Coupled { Poisson Electron Hole }

    #-b) ramp drain to positive starting voltage
    Quasistationary (
        InitialStep=0.1 MaxStep=0.5 Minstep=1.e-5
        Goal { Parameter=vd.dc Voltage=2 }
    )
    { Coupled { Poisson Electron Hole } }

    #-c) ramp gate to negative starting voltage
    Quasistationary (
        InitialStep=0.1 MaxStep=0.5 Minstep=1.e-5
    )
}
```

```

Goal { Parameter=vg.dc Voltage=-2 }
)
{ Coupled { Poisson Electron Hole } }

#-d) ramp gate -2V..3V : AC analysis at each step.
Quasistationary (
    InitialStep=0.01 MaxStep=0.04 MinStep=1.e-5
    Goal { Parameter=vg.dc Voltage=3 }
)
{ ACCoupled (
    StartFrequency=1e6 EndFrequency=1e6
    NumberOfPoints=1 Decade
    Node(d s g b) Exclude(vd vs vg vb)
)
{ Poisson Electron Hole }
}
}

```

The initial solution is obtained in steps. It starts with the Poisson equation and introduces the electron and hole carrier continuity equations, and the circuit equations, which are included by default.

```

#-d) ramp gate -2V to +3V : AC analysis at each step.
Quasistationary (
    InitialStep=0.01 MaxStep=0.04 MinStep=1.e-5
    Goal { Parameter=vg.dc Voltage=3 }
)
{ ACCoupled (
    StartFrequency=1e6 EndFrequency=1e6
    NumberOfPoints=1 Decade
    Node(d s g b) Exclude(vd vs vg vb)
)
{ Poisson Electron Hole }
}

```

This third Quasistationary statement performs an AC analysis at a single frequency (1×10^6 Hz) at each DC bias step during a sweep of the voltage source vg, which is attached to the gate.

Analysis at multiple frequencies is possible by defining a value for StartFrequency and EndFrequency.

`Node(d s g b)`

The AC analysis is performed between the circuit nodes d, s, g, and b. The conductance and capacitance matrices contain 16 elements each: a(d,d), c(d,d), a(d,s), c(d,s), ..., a(b,b), c(b,b).

1: Overview

Example: Small-Signal AC Extraction

Exclude (vd vs vg vb)

Excludes all voltage sources from the AC analysis.

Results of AC Simulation

When Sentaurus Device is run inside the Sentaurus Workbench project advanced_AC, the simulation results are plotted automatically after completion by Inspect, which is driven by the command file pp3_ins.cmd. The total small-signal gate capacitance $C_g=c(g,g)$ is plotted against gate voltage. Two small-signal conductances are also plotted (see Figure 14), where $g_m=a(d,g)$ is the small-signal transconductance and $g_d=a(d,d)$ is the small-signal drain output conductance.

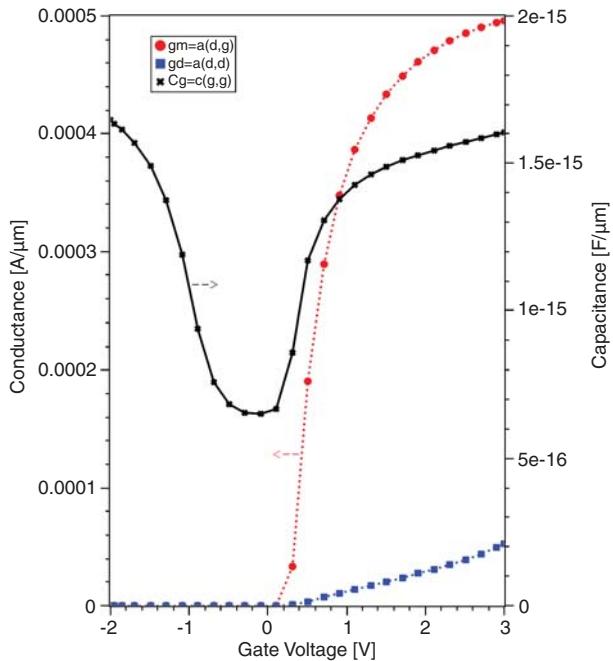


Figure 14 Inspect output from Sentaurus Workbench project AC simulation

This chapter describes the commands required for single-device simulations.

Overview

The Sentaurus Device command file is divided into sections that are defined by a keyword and braces (see [Figure 15](#)). 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 141](#). This part of the manual concentrates on the input to define single device simulations.

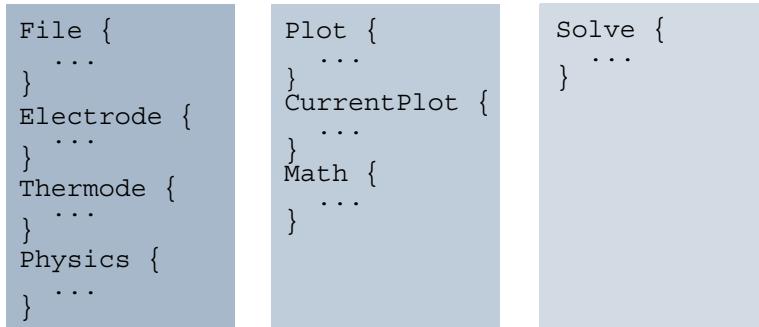


Figure 15 Different sections of a Sentaurus Device input file

Specifying the Device

A device is defined by its mesh and doping (File section), contacts and thermodes (Electrode and Thermode sections), and the physical models that are selected. Physical models can be selected globally or for specific materials, regions, or interfaces (in various Physics sections).

Defining the Output

Device simulations generate substantial data. For a given simulation, the results to be saved are specified in the Plot and CurrentPlot sections.

Specifying the Simulation

With any structure, such as a device, different simulations are possible. These are defined in the `Solve` section, and parameters for the methods used are defined in the `Math` section.

Inserting Files

An insert directive is available in the command file of Sentaurus Device to incorporate other files:

```
Insert = "filename"
```

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
- Thermode

The following search strategy is used to locate a file:

- The current directory is checked first (highest priority).
- If the environment variable `SDEVICEDB` exists, the directory associated with the variable is checked (medium priority).
- Finally, the `$STROOT/tcad/$STRELEASE/lib/sdevice/MaterialDB` directory is checked (lowest priority).

NOTE The insert directive is also available for parameter files (see [Library of Materials on page 117](#)).

File Section

File names for a simulation are specified in the `File` section. Each command uses a predefined file extension. Therefore, file names are given without extensions. [Table 127 on page 1091](#) lists all the parameters with which to specify file names.

In most cases, a device can be specified using only the `Grid` and `Doping` files. Where a simulation depends on previous results, the `Load` command loads a previously computed solution. The `Parameters` file is used to change the standard model parameters for a device.

The outputs of a device simulation can be saved for reuse (`Save` file) or plotting (`Plot` file). The voltage, charge, and current values at the electrodes can be logged (`Current` file). The `Plot` file contains a default set of variables and any variables specified by you in the `Plot` section. The `Save` file contains only data that is necessary to restart the simulation after reading it from a `Load` file.

The `Output` file receives log messages of Sentaurus Device. While this information is of little importance from an application perspective, it is often useful to diagnose problems (for example, with convergence).

The keywords `Compressed`, `SaveCompressed`, `PlotCompressed`, and `CurrentCompressed` can be used to obtain compressed output files and reduce disk space use.

Other files are more specific and are discussed in context in other chapters.

Instead of separate DF-ISE files for the geometry and the doping profiles, a single TDR file can be specified, for example:

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

No doping file needs to be specified in this case. If the grid file is in the TDR format, Sentaurus Device will also use the TDR format for save and plot files.

2: Basic Sentaurus Device

Electrode Section

Electrode Section

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

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

By default, contacts are Ohmic or gate contacts (see [Electrical Boundary Conditions on page 206](#)). [Table 146 on page 1110](#) lists the additional keywords that can be specified for each electrode.

The parameters `Charge` and `Current` are conceptually different from the other parameters in that they determine the boundary condition type for an electrode (which is related to an experimental setup), while the other parameters describe physical properties of the electrodes (that relate to the device itself). 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. It is possible to change the boundary condition for an electrode during the simulation (see [Quasistationary Command on page 67](#) and [Set Command on page 91](#)). 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.

Parameters such as `Voltage`, `Current`, and `Charge` can be written with multiple values using the syntax [`<float>, <float>, ... , <float>`]. The simulation is run as many times as values are specified. Most of the options, except `Barrier`, `AreaFactor`, and `eRecVelocity` or `hRecVelocity`, cannot be used with other options.

`AreaFactor` is 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 explore 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 `Physics` section, with the same meaning; if both `AreaFactors` are present, Sentaurus Device multiplies them.

For 1D or 2D devices, the parameters Resist, Current, and FGcap are not absolute values. For example, Current specifies a current density rather than a current. Therefore, if the measured values R [Ω], I [A], and C [F] are known, then the electrode values should be computed as follows:

- Resist=R*AreaFactor
- Current=I/AreaFactor
- FGcap=C/AreaFactor

For example, if a 5Ω contact resistance is experimentally measured for a $10\text{ }\mu\text{m}$ wide contact, AreaFactor=10 and Resist=50 ($5 \times 10\text{ }\mu\text{m}\Omega$) are used.

When distributed contact resistance is specified, each node of the electrode has a separate resistor with a resistance proportional to $1/A_i$, where A_i is the area associated with the node.

Connections to circuit nodes must be resistive. If an electrode is not specified as being resistive and is connected to a circuit node, Sentaurus Device converts the electrode into a resistive contact (with a default value of 0.001Ω). In some applications, a 0.001Ω resistor can influence simulation results, in which case an explicit resistor definition using the keyword Resist is recommended.

Workfunction and Material Specifications for Contacts

Sentaurus Device supports the specification of a workfunction or material for an electrode instead of using the Barrier definition. This is useful in two important cases: electrodes without any contact to semiconductors (such as the gate in a MOSFET) and Schottky contacts on semiconductors.

The workfunction is specified using the syntax Workfunction = <value> [eV] in the Electrode section, for example:

```
Electrode {
    { Name = "gate"  Voltage = 0.0  Workfunction = 4.15 }
}
```

To use a default workfunction of a material (or from the parameter file), the material name can be specified: Material = "Name". For electrodes such as a MOSFET gate, it is useful to specify both the semiconductor material and doping concentration (for example, for polysilicon gates). In this case, the syntax is:

- Material = "Name" (N = <value>) for n-type impurities
- Material = "Name" (P = <value>) for p-type impurities

2: Basic Sentaurus Device

Electrode Section

For example:

```
Electrode {  
    { Name = "gate" Voltage = 0.0 Material = "Silicon"(P=7.5e19) }  
}
```

The built-in potential is approximated by the standard expression $kT\ln(N/n_{i,eff})/q$. If the Fermi level is required to be equal to the conduction or valence band energy, the doping specification must be omitted. For example, for an n⁺-polysilicon gate:

```
Electrode {  
    { Name = "gate" Voltage = 0.0 Material = "PolySi"(N) }  
}
```

NOTE Using the Workfunction or Material specification is mandatory for Schottky electrodes that contact several different semiconductors. In such a case, the Barrier specification gives the same barrier to each semiconductor, which is incorrect.

NOTE If Material or WorkFunction is specified for an Ohmic contact that is in contact with both the semiconductor and insulator, the electrostatic potential equals the built-in potential at semiconductor nodes, but for insulator nodes, it corresponds to the electrode Workfunction.

Table 1 Various electrode declarations

Description	Command statement
Gate electrode	{ name = "gate" voltage = 2 Material = "PolySi"(P) }
Schottky electrode	{ name = "anode" voltage = 2 Schottky WorkFunction = 4.9 }
Current boundary	{ name = "anode" voltage = 1.0 current = 1e-3 }
Floating electrode with charge	{ name = "floatgate" charge = 1e-15 }
Electrode with area factor	{ name = "anode" voltage = 2 AreaFactor=100 }
Electrode with 1 Ω resistance	{ name = "emitter" voltage = 2 Resist=1 }
Multivalued voltage drive	{ name = "source" voltage = [0, 1.0, 2.0] }
In the above example, all commands of the Solve statement are executed with an initial 0 source voltage. Then, they are repeated with "source" voltage = 1 V, then with "source" voltage = 2 V.	
Time-dependent voltage drive	{ name = "source" voltage = (0 at 0, 0.2 at 1e-6, 0.5 at 2e-6) }
In the above example, the source assumes these voltages during a transient simulation: 0 V at 0 s, 0.2 V at 1 μs, and 0.5 V at 2 μs.	

Table 1 Various electrode declarations

Description	Command statement
Time-dependent voltage drive	{ name = "source" voltage = 0 voltage = (5 at 0, 10 at 1e-6) }

In the above example, both the initial voltage (0 V) and time-dependent values (5 V at 0 s, 10 V at 1 ms) have been specified for the source. This combination is useful where an initial quasistationary command ramps the source voltage from 0 V to 5 V, before the source voltage increases to 10 V during a subsequent transient analysis.

Thermode Section

The Thermode section defines the thermal contacts of a device.

The Thermode section is defined in the same way as the Electrode section. Each thermode is defined in a section between braces and must include a name and default temperature, for example:

```
{ Name = "thermode1" Temperature=300 }
```

A complete Thermode section can be:

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

[Table 148 on page 1111](#) lists the keywords available for the Thermode section.

Table 2 Various thermode declarations

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

2: Basic Sentaurus Device

Physics Section

Physics Section

The Physics section is used to select the models that are used to simulate a device. [Table 159 on page 1125](#) lists the keywords that are available, and Part II and Part III discuss the 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. Other specifications are syntactically possible everywhere, but are valid for certain materials only; for example, the Traps model is semantically valid only for semiconductors, and the Charge model is valid only for insulators.

The two uses of specification in the Physics section are:

- Commands for model selection (for example, Mobility, Recombination).
- Modifiers of parameters for global models (for example, Thermodynamic, Hydrodynamic).

Physics Section Example

This example illustrates many of the possible keywords available in the Physics section:

```
Physics {
    Temperature=300
    # bandgap narrowing on:
    EffectiveIntrinsicDensity (BandGapNarrowing (BennettWilson ))
    Mobility (
        DopingDependence
        # default model:
        CarrierCarrierScattering(ConwellWeisskopf)
        HighFieldSaturation # default GradQuasiFermi
        Enormal
    )
    Recombination (
        # no trap-assisted tunneling:
        SRH ( DopingDependence )
        Auger
        TrapAssistedAuger
        # using non-default driving force, but default model:
```

```
Avalanche(vanOverstraeten Eparallel)
Band2Band
)
AlphaParticle (
    Energy=5e6 Time=0 # default time
    Direction = (1,2)
    Location  = (5,0)
)
}
```

Region-specific and Material-specific Physics

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>
}
```

For example, different charges can be defined in different insulator regions by specifying the Charge(conc = <number>) statement (unit is cm^{-3}) in the Physics section of the appropriate region.

A Physics section without any region or material specifications is considered the default section. The hierarchy of region or material Physics and Math specifications is presented in [Model Hierarchy on page 121](#).

2: Basic Sentaurus Device

Physics Section

NOTE Region names can be edited in Sentaurus Structure Editor (see [Sentaurus Structure Editor User Guide, Changing the Name of a Region on page 110](#)).

Regionwise or materialwise specification is not allowed for these models:

AnalyticTEP	Fermi	Hydrodynamic	MagneticField	Piezo
RecGenHeat	SiC	Temperature	Thermodynamic	

See [Specifying Model Parameters on page 115](#).

Physics at Interfaces

A special set of models can be activated at the interface between two different materials or two different regions. In [Table 159 on page 1125](#), pure interface models are flagged with '(i)' in the description column.

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 Model Syntax

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") {
    Charge(Conc=-1.e12)
}
```

Interface Model Parameters

If a model is defined in an interface-specific Physics section, the parameters of that model must be specified in the parameter file in a parameter set for the same interface. For example, for the statement:

```
Physics (MaterialInterface="Oxide/Silicon") {
    Recombination(surfaceSRH)
}
```

the corresponding model parameters must be defined in the parameter file of Sentaurus Device:

```
MaterialInterface="Oxide/Silicon"{
    SurfaceRecombination * surface SRH recombination:
    { # S0 = 1e3 , 1e3 # [cm/s]
        S0 = 100 , 100 # [cm/s] }
}
```

This parameter definition also applies to:

```
Physics (RegionInterface="Region.0/Region.1") {
    Recombination(surfaceSRH)
}
```

if the Region.0/Region.1 interface is an oxide–silicon interface.

Physics at Electrodes

Electrode-specific Physics sections can be defined, for example:

```
Physics(Electrode="Gate"){
    Schottky
    eRecVel = <value>
    hRecVel = <value>
    Workfunction = <value>
}
```

Plot Section

The `Plot` section specifies the data that is saved by the `Plot` command in the `File` section or by the `Plot` command in the `Solve` section (see [Plot, Save, and Load Commands on page 84](#)). Refer to [Table 225 on page 1180](#) for all possible plot options.

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
}
```

These quantities do not have to be defined in the `Physics` section to be saved in a plot file. The plot file saves only the current data generated by the simulation.

Interface Plots

Data fields defined on interfaces can be plotted by using the modifier `/RegionInterface`:

```
Plot {
    HotElectronInj/RegionInterface
}
```

The following fields are available for interface plots:

```
SurfaceRecombination
HotElectronInj
HotHoleInj
```

NOTE These fields can also be plotted on regions by omitting the qualifier `/RegionInterface`. However, they are zero inside bulk regions, and nonzero values are only produced for vertices along interfaces.

Interface plots are only generated if interface regions appear in the grid file. If necessary, interfaces can be added to existing DF-ISE grid and data files by executing the command:

```
dfisetools -aid input.grd input.dat output.grd output.dat
```

For TDR files, the following command is available:

```
snmesh -u -AI input.tdr
```

In 2D, interface plots can be visualized as follows in Tecplot SV:

1. In Tecplot SV, **Plot > Zone/Mapping Style** for the required interfaces.
2. In the **Zone Style** dialog box:
 - a) Click **Edge** and set Show Edges to No.
 - b) Click **Mesh**, and set Mesh Show to Yes, set Mesh Color to Multi C1, and increase Line Thck as required to improve the visibility.
3. Click **Close**.

The mesh color will reflect the values of the field.

In 3D, interface plots can be visualized as follows in Tecplot SV:

1. In Tecplot SV, **Plot > Zone/Mapping Style** for the required interfaces.
2. In the **Zone Style** dialog box, click **Contour** and set Cont Show to Yes.
3. Click **Close**.

The surface color will reflect the values of the field.

CurrentPlot Section

The CurrentPlot section is used to include selected mesh data into the current plot file (.plt). The same variables can be selected as in the Plot section (see [Appendix F on page 1059](#)).

Data can be plotted according to node numbers or coordinates. The node numbers are obtained by probing the mesh using Tecplot SV (see [Tecplot SV User Guide, Environment Variables on page 48](#)).

Node numbers are given as plain integers. However, a coordinate is given as one to three (depending on device dimensions) numbers in parentheses, which distinguish coordinates from node numbers. When plotting according to coordinates, the plotted values are interpolated as required.

Furthermore, it is possible to output averages, integrals, and the maximum and minimum of quantities over specified domains. To do this, specify the keyword Average, Integrate,

2: Basic Sentaurus Device

CurrentPlot Section

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=<regionname>`
- Material specification: `Material=<materialname>`
- Region interface specification: `RegionInterface=<regioninterfacename>`
- Material interface specification: `MaterialInterface=<materialinterfacename>`
- 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)`

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=<plotname>` 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.)

In addition to the average, integral, maximum, and minimum over the specified domain, Sentaurus Device has the option to output, in the `.plt` file, the coordinates where average, integral, maximum, and minimum occur. In the case of average and integral, Sentaurus Device computes the coordinates ($\langle x \rangle$, $\langle y \rangle$, $\langle z \rangle$) of the centroid of a data field f defined as:

$$\langle x \rangle = \frac{\int xf(x, y, z)dV}{\int f(x, y, z)dV} \quad (2)$$

where the integration covers the specified domain. The values of $\langle y \rangle$ and $\langle z \rangle$ are defined similarly. This option 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 one-dimensional, 2D, or 3D window is defined by the coordinates (in micrometers) of two opposite corners of the window. In addition, it is possible to 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 domains are neglected if the window or well is a valid one. If the keyword `Coordinates` is also used in the domain-specification parentheses, then the coordinates where average, integral, maximum, and minimum occur inside the window or well are output to the `.plt` file.

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. [Ramping Physical Parameter Values on page 71](#) describes how model names and parameter names can be determined. Finally, Sentaurus Device also provides a current plot PMI (see [Current Plot File of Sentaurus Device on page 1027](#)).

NOTE Do not confuse the CurrentPlot section with the CurrentPlot statement in the Solve section introduced in [CurrentPlot Section on page 88](#).

Example: Node Numbers

Consider this device declaration in a Sentaurus Device input file:

```
Device CAP {
    Electrode { { Name = "Top" Voltage = 0.0 }
                 { Name = "Bot" Voltage = 0.0 } }

    File { grid = "cap_mdr"
           doping = "cap_mdr" }

    Plot { Potential ElectricField/Vector SpaceCharge
           eDensity eCurrent/Vector eQuasiFermi
           hDensity hCurrent/Vector hQuasiFermi }

    CurrentPlot { Potential (7, 8, 9)
                  ElectricField (7, 8, 9) }
}
```

In addition to the usual contact currents, the current file of the device CAP contains the electrostatic potential and electric field for the mesh vertices 7, 8, and 9.

2: Basic Sentaurus Device

CurrentPlot Section

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 (7, 8, 9) } }
    ...
}
```

Example: Advanced Options

This example is a 2D device that uses the more advanced CurrentPlot features:

```
CurrentPlot {
    eDensity (0 1) * plot electron density at nodes 0 and 1
    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
        * material specification is redundant, therefore the same as above will
        * be plotted:
        Maximum(Semiconductor Material="Silicon")
        * minimum in a material and a region, output under the name "x":
        Minimum(Name="x" Material="Oxide" Region="Channel")
        * minimum in a region and all insulator regions:
        Minimum(Region="Channel" Insulator)
    )
    eDensity(
        * maximum and coordinates of maximum over semiconductors
        Maximum(Semiconductor Coordinates)
        * average and coordinates of centroid
        Average(Semiconductor Coordinates)
        * minimum and coordinates of minimum over semiconductors
        Minimum(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)])
    * average and average coordinates in window
    Average(Window[(-0.1 0.3) (0.2 0.2)])
    * integral over 2D window
    Integrate(Window[(-0.1 0.3) (0.2 0.2)])
    * integral over well
    Integrate( DopingWell(-0.1 0.3) )
)
}
```

Example: Physical 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"
}
```

TaileDistributionPlot and TailhDistributionPlot Sections

This section is used to plot tail electron distribution and tail hole distribution versus energy at specified positions (see [Visualizing Tail Distribution on page 544](#)).

TrappedCarDistrPlot Section

This section is used to plot trapped carrier density and occupancy probability versus energy at specified positions. This trap-related plotting command is described in [Visualizing Traps on page 359](#).

2: Basic Sentaurus Device

NonLocalPlot Section

NonLocalPlot Section

The NonLocalPlot section is used to visualize data defined on nonlocal lines (see [Visualizing Data Defined on Nonlocal Meshes on page 106](#)).

Solve Section

The `Solve` section is the only section in which the order of commands and their hierarchy 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 input file. Many `Solve` commands are high-level commands that have lower level commands as parameters. [Figure 16](#) shows an example of these different command levels:

- The `Coupled` command (the base command) is used to solve a set of equations.
- The `Plugin` command is used to iterate between a number of coupled equations.
- The `Quasistationary` command is used to ramp a solution from one boundary condition to another.
- The `Transient` command is used to run a transient simulation.

Furthermore, small-signal AC analysis can be performed with the `ACCoupled` command. An advanced ramping by continuation method can be performed with the command `Continuation`. (The `ACCoupled` and `Continuation` commands are presented in [ACCoupled: Small-Signal AC Analysis on page 161](#) and [Continuation Command on page 75](#), respectively.)

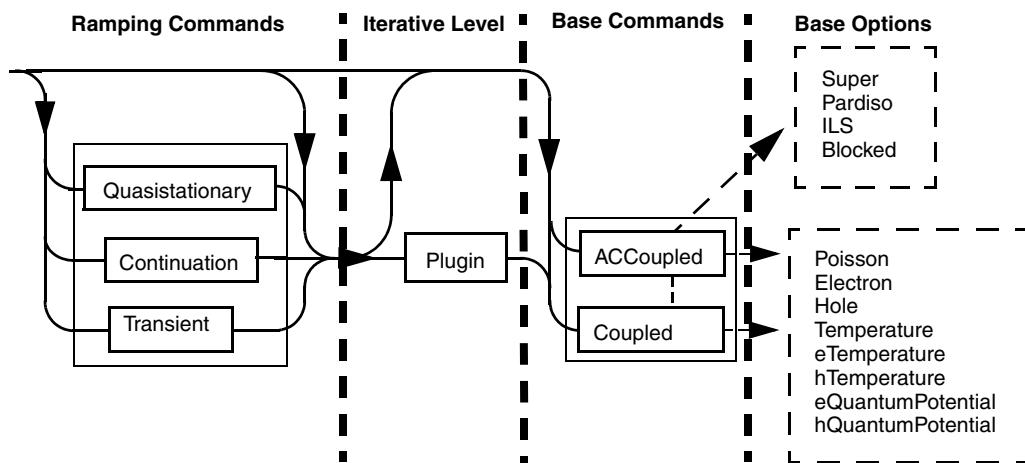


Figure 16 Different levels of Solve commands

Coupled Command

The Coupled command activates a Newton-like solver over a set of equation–variable pairs. In Sentaurus Device, the basic semiconductor model equations are the Poisson equation, the two continuity equations, and the different thermal and energy equations. [Table 131 on page 1099](#) presents a list of the keywords used in a Coupled command.

The syntax of the Coupled command is:

```
Coupled ( <optional parameters> ){ <equation-variables> }
```

or equivalently:

```
<equation-variable>
```

This last form uses only the keyword `equation-variable`, which is equivalent to a coupled with default parameters and the single equation–variable. For example, if the following command is used:

```
Coupled { Poisson Electron }
```

the electrostatic potential and electron density are computed from the resolution of the Poisson equation and electron continuity equation (using the default parameters).

If the following command is used:

```
Poisson
```

only the electrostatic potential is computed using the Poisson equation.

The Coupled command is based on a Newton solver. This is an iterative algorithm in which a linear system is solved at each step simulation. The possible parameters of the command are:

- The maximum number of iterations allowed.
- The desired 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 are summarized in [Table 131 on page 1099](#). The command is controlled by both an absolute criterion and a relative error criterion. The relative error control can be specified with the optional parameter `Digits`. The absolute error control can be specified in the Math section (see [Math Section on page 91](#)).

2: Basic Sentaurus Device

Solve Section

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. In this context, allow the Newton algorithm to proceed as far as possible. Inside a ramping command (for example, Quasistat, Transient), the maximum number of iterations must 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 linear method used in a coupled iteration depends on the type and size of the problem solved. [Table 154 on page 1123](#) lists the solvers and the size of the problems for which they are designed.

Line search damping (option LineSearchDamping) and Bank–Rose damping (option NotDamped) are unrelated damping methods. Unlike the right-hand side that Sentaurus Device shows in the solve report, line search damping is based on the Fedorenko norm. Therefore, despite damping, the right-hand side in the solve report can increase.

Plugin Command

The Plugin command controls an iterative loop over two or more Coupled commands. 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 ( <optional parameters> ) ( <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 {  
    Coupled { Poisson }  
    Coupled { Electron }  
    Coupled { Hole }  
}
```

or using the abbreviated Coupled command, as:

```
Plugin { Poisson Electron Hole }
```

As the `Plugin` command loops through several `Coupled` commands, it takes as its options:

- The maximum number of iterations to be performed.
- The required precision of the result.
- The capability to stop the iterative process if an inner `Coupled` does not converge.

[Table 138 on page 1105](#) lists the corresponding keywords of these parameters.

`Plugin` commands can be used with other `Plugin` commands, such as:

```
Plugin{ Plugin{ ... } Plugin { ... } }
```

[Figure 17](#) illustrates the corresponding loop structure. A hierarchy of `Plugin` commands allows more complex iterative solve patterns to be created.

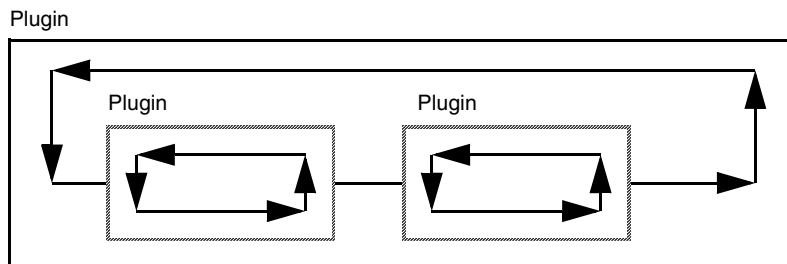


Figure 17 Example of hierarchy of `Plugin` commands

Quasistationary Command

The `Quasistationary` command is used to ramp a device from one solution to another through the modification of its boundary conditions (for example, ramping the voltage at a contact) 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 18](#)). The command to re-solve the device at each iteration is given with the `Quasistationary` command.

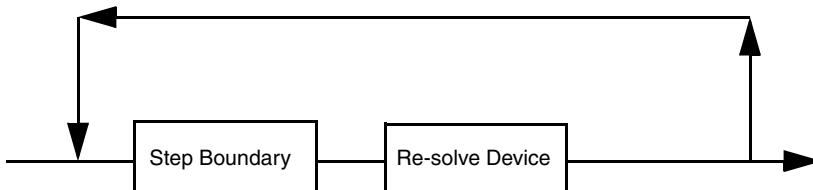


Figure 18 Structure of `Quasistationary` command

2: Basic Sentaurus Device

Solve Section

Ramping Boundary Conditions

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

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

The possible parameters are listed in [Table 139 on page 1105](#). In the above command, `<solve-command>` is Coupled, Plugin, 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. This means that the corresponding 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.

Greater control of the command is possible with the Step control parameters that affect the behavior 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 `InitialStep`, `MaxStep`, `MinStep`, `Increment`, and `Decrement`:

- `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.
- `MaxStep` and `MinStep` limit this change.
- The rate of increase is controlled by the number of performed Newton iterations and by the factor `Increment`.
- The step size is only reduced by the factor `Decrement`, when the inner solve fails. Consequently, the ramping process stops when the `MinStep` condition is reached.

As some of the control of the Quasistationary command is through an internal variable t , some conversion may be necessary. The relation is linear. For example, if a contact has an initial value of 2 V and the goal of the Quasistationary command is 5 V on this contact, an `InitialStep` of 0.1 corresponds to a 0.3 V step on the contact. This conversion process is often necessary to specify when to plot data during the ramping.

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. The goal and contact type

must match. If they do not match, Sentaurus Device changes the contact type to match the goal. If the goal is current and the contact type is voltage, Sentaurus Device changes the contact type to current. If the goal is voltage and the contact type is current, Sentaurus Device changes the contact type to voltage. If the goal is voltage and the contact type is charge, the contact type is changed to voltage. If the goal is charge and the contact type is voltage, the contact type is changed to charge. However, Sentaurus Device cannot change a contact of charge type to current type, or a contact of current type to charge type.

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 [Set Command on page 91](#)).

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 (CIS) and charge-coupled device (CCD) simulations.

To activate the model, 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>] }
```

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 code above).

When the semiconductor well where the quasi-Fermi potential to be biased is connected to a contact, the WellContactName specification can be used as well. Now, the contact associated with the well must be specified to select the well (see code above). This option cannot be used when the doping well is buried (with no associated contact).

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}}
}
```

2: Basic Sentaurus Device

Solve Section

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

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

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=<region> | Material=<material> | Semiconductor])
    eQuasiFermi=5.0)
```

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

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. 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 115](#)).

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 3](#) contain command file parameters that can be ramped.

Table 3 Command file parameters

Model name	Parameters
OptBeam(<index>) RayBeam(<index>)	RefractiveIndex, SemAbs, SemSurf, SemVelocity_Vx, SemVelocity_Vy, SemVelocity_Vz, SemWindow_xmax, SemWindow_xmin, SemWindow_ymax, SemWindow_ymin, SemWindow_zmax, SemWindow_zmin, WaveDir_x, WaveDir_y, WaveDir_z, WaveEnergy, WaveInt, WaveLength, WavePower, WaveTime_tmax, WaveTime_tmin, WaveTsigma, WaveXYsigma
OpticalGeneration(<index>)	WaveLength
Optics	Wavelength, Amplitude, WavePower, WaveIntensity, Theta, Phi, Polarization
RadiationBeam	Dose, DoseRate, DoseTSigma, DoseTime_end, DoseTime_start

2: Basic Sentaurus Device

Solve Section

Table 3 Command file parameters

Model name	Parameters
Traps(<index>)	Conc, EnergyMid, EnergySig, eGfactor, eJfactor, exsection, hGfactor, hJfactor, hxsection
DeviceTemperature	Temperature

NOTE Certain models such as optical beams and traps must be specified with an index:

```
Model="OptBeam(0)"  
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 4](#) can appear in a Goal statement.

Table 4 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 $X_{max}(0), X_{max}(1), \dots$
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 $[X_{max}(0), X_{max}(1)], [X_{max}(1), X_{max}(2)], \dots$

Mole fraction-dependent parameters can be ramped in all materials. In mole fraction-dependent materials, the interpolation values $p(\dots)$ and the 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, Sentaurus Device issues a warning, and the corresponding goal statement is ignored.

Parameters in PMI models can also be ramped.

Saving and Plotting Data During Quasistationary Solve Sequence

Data can be saved and plotted during a Quasistationary ramping process by using the `Plot` command. The `Plot` option 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.

Another way to plot data is with the `Plot` statement in the `Solve` command rather than inside the `Quasistationary` statement (see [Plot, Save, and Load Commands on page 84](#)). This command is added after the given `Solve` command, 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 )
}
```

Extrapolation

If the `Extrapolate` option is present in the `Math` section, the `Quasistationary` command uses linear extrapolation of the last two solutions to predict the next solution. This extrapolation information is preserved between `Quasistationary` commands, and it is also saved and loaded automatically by the `Save` and `Load` commands.

A `Quasistationary` command can use 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 }}
```

2: Basic Sentaurus Device

Solve Section

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

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 Quasistationary options in [Table 5](#) control the handling of extrapolation information.

Table 5 Extrapolation options

Option	Description
ReadExtrapolation	Tries to use the extrapolation information from a previous command if it is available and compatible. This is the default.
-ReadExtrapolation	Do not use the extrapolation information from a previous command.
StoreExtrapolation	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.
-StoreExtrapolation	Do not store the extrapolation information internally at the end of the command.

Continuation Command

The Continuation command enables automated tracing of arbitrarily shaped $I(V)$ curves of complicated device phenomena such as breakdown or latchup. Simulation of these phenomena usually requires biasing conditions tracing a multivalued $I(V)$ curve with abrupt changes. The implementation is based on a dynamic load-line technique [1] adapting the boundary conditions along the traced $I(V)$ curve to ensure convergence. An external load resistor is connected to the device electrode at which the $I(V)$ curve is traced, the device being indirectly biased through the load resistance. The boundary condition consists of an external voltage applied to the other end of the load resistor not connected to device. By monitoring the slope of the traced $I(V)$ curve, an optimal boundary condition (external voltage) is determined by adjusting the load line so it is orthogonal to the local tangent of the $I(V)$ curve. The boundary conditions are generated automatically by the algorithm without prior knowledge of the $I(V)$ curve characteristics.

An important part of the continuation method consists of computing the slope in each point of the $I(V)$ curve. This is equivalent with computing the inverse of the device differential resistance when a small-voltage perturbation is applied to the contact undergoing continuation. The simulation advances to the next operating point if the solution has converged. Before moving to the next step, the load line is recalibrated so that it is orthogonal to the local tangent of the $I(V)$ curve. This ensures an optimal boundary condition. A user-defined window specifies the limits for curve tracing. The tracing window is specified by user-defined lower and upper values for the voltage and current of the operating point at the continuation electrode: MinVoltage, MaxVoltage, MinCurrent, and MaxCurrent. The simulation ends when the operating point is outside the tracing window.

The continuation method is activated by using the keyword Continuation in the Solve section. Some control parameters must be given in parentheses in the same way as for the Plugin statement, for example:

```
Continuation (<Control Parameters>) {  
    Coupled (iterations=15) { poisson electron hole }  
}
```

[Table 130 on page 1098](#) 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.

The first step of the continuation is always a voltage-controlled step. For this, you must supply an initial voltage step in the control parameter list using the InitialVstep statement. The continuation proceeds automatically until the values given by any of the MinVoltage, MaxVoltage, MinCurrent, or MaxCurrent parameters are exceeded. In addition, the parameters Increment, Decrement, Error and Digits can be specified. Their definitions

2: Basic Sentaurus Device

Solve Section

are the same as for Transient and Quasistationary statements except that they measure the $I(V)$ curve arc length.

In the regions where the $I(V)$ curve becomes vertical (close to current boundary condition) and the current extends over several orders of magnitude for almost the same applied voltage, another parameter, `MinVoltageStep`, may need to be adjusted. `MinVoltageStep` is the minimum-allowed voltage difference between two adjacent operating points on the $I(V)$ curve. In such cases, increasing the parameter value produces a smoother curve over the vertical range.

Tracing successfully an $I(V)$ curve with the continuation method depends on how accurately the local slope of the traced $I(V)$ curve is computed. As slope computation involves using inverted Jacobian and current derivatives at the electrode, an accurate computation of the contact current and a low numeric noise in Jacobian are prerequisites for continuation.

At low-biasing voltages, current at the continuation electrode is very small and, in general, noisy. This leads to an inaccurate computation of the slope, which in turn causes the continuation method to backtrace. To overcome this problem, Sentaurus Device allows you to divide the continuation window into two regions separated by a threshold current with a value specified by the parameter `Iadapt`. The lower region is a low-current range where the slope computation is inaccurate, and the upper region is now the region where the continuation method is expected to work as designed.

From `MinCurrent` to `Iadapt` (the lower region of the simulation window), the adaptive algorithm is switched off and a fixed-value resistor is used instead. In this range, the simulation proceeds as a voltage ramping through a fixed-value resistor attached to the continuation electrode. Because no slope computation is necessary, the sensitivity of the simulation to current noise is eliminated. When the current increases to the value specified by `Iadapt`, the adaptive continuation is switched on and the curve tracing proceeds as previously described. The default value for the fixed resistor used in the lower region is 0.001Ω , and it can be changed by either using the continuation parameter `Rfixed` or specifying the `Resist` keyword in the `Electrode` section of the continuation electrode.

An example of a `Solve` entry for continuation is:

```
Electrode{  
    ...  
    {Name="collector" Voltage=0.0}  
}  
  
Solve{ ...  
    Continuation ( Name="collector" InitialVstep=-0.001  
        MaxVoltage=0 MaxCurrent=0  
        MinVoltage=-10 MinCurrent=-1e-3  
        Iadapt=-1e-13) {
```

```

        Coupled (iterations=10) { poisson electron hole }
    }
}
}
```

This specifies that the $I(V)$ curve must be traced at the electrode "collector". The initial voltage-controlled step is -0.001 V , the voltage range is -10V to 0V , and the current range is -1mA to 0A . The adaptive algorithm is switched on when the current reaches $-1.0 \times 10^{-13}\text{ A}$ to avoid low-current regime noise.

The step along the traced $I(V)$ curve is controlled primarily by the convergence. The step increases by a factor `Increment` if the problem has converged. When the problem does not converge, the step is cut by a factor `Decrement` and the problem is re-solved. The step cut continues until the problem converges. The continuation parameters `Increment` and `Decrement` are available for adjustment in the `Continuation` section, and the default values are 2.0 for `Increment` and 1.5 for `Decrement`.

Sentaurus Device also allows a more complex step control based on both convergence and curve smoothness. This is activated by specifying the keyword `Normalized` in the `Continuation` section. The angle between the last two segments on the $I(V)$ is computed in a local scaled I-V plane, with the scaling factors depending on the current point on the $I(V)$ curve. If the angle is smaller than the continuation parameter `IncrementAngle`, the step increases by the factor `Increment`. If the angle is greater than `IncrementAngle` but smaller than `DecrementAngle`, the step is kept constant. Finally, if the angle is greater than `DecrementAngle`, the step decreases by the factor `Decrement`. Default values for `IncrementAngle` and `DecrementAngle` are 2.5 and 5 degrees, respectively.

A few more options are available for limiting the step (arclength) along the traced $I(V)$ curve. By specifying the continuation parameter `MaxVstep`, the step along the $I(V)$ curve is limited to an upper value such that the step projection on the V-axis is smaller in absolute value than `MaxVstep`. This option is particularly useful for a low-voltage range when a curve with more points is needed. In the higher-current range, one of the continuation parameters `MaxIstep`, `MaxIfactor`, or `MaxLogIfactor` can be used to limit the step along the curve. When `MaxIstep` is specified, the step is limited such that the step projection on the I-axis is smaller in absolute value than `MaxIstep`. `MaxIfactor` specifies how many times the current is allowed to increase for two adjacent points on the $I(V)$ curve. Similarly, `MaxLogIfactor` specifies by how many orders of magnitude the current is allowed to increase for two adjacent points on the $I(V)$ curve. `MaxVstep` can be combined with one of `MaxIstep`, `MaxIfactor`, or `MaxLogIfactor`.

In mixed mode, the continuation method allows you to have all other contacts except the continuation contact connected in a circuit. The continuation contact should not be connected to any circuit node. In this case, the `Name` keyword in the `Continuation` section specifying the continuation contact name is replaced by `dev_inst.Name`, where `dev_inst` represents the device instance of the respective contact. For example, for device `mos1` with electrodes

2: Basic Sentaurus Device

Solve Section

named source, drain, gate, and substrate undergoing continuation on the drain electrode, the possible syntax is:

```
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}}
}
```

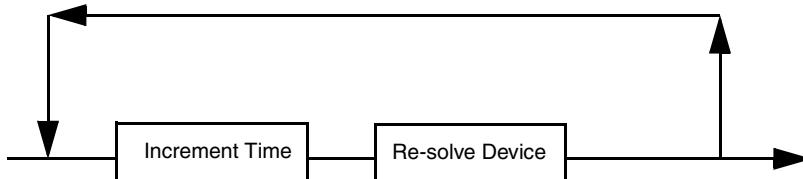
In mixed-mode simulations, sometimes the continuation electrode must be biased to a certain voltage using a Quasistationary simulation before the continuation starts. In mixed mode, because the continuation electrode is not allowed to be connected to any node, special syntax must be used to avoid biasing the contact as a node. For example, in the above syntax, the drain electrode needs to be biased to 3 V before the continuation. Instead of Name, the Contact keyword is used to identify the drain contact:

```
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 19](#)). The command to solve the device at each iteration is given with the Transient command.



[Figure 19](#) Structure of Transient command

The syntax of the Transient command is:

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

[Table 141 on page 1107](#) lists the possible parameters.

In the above command, `<solve-command>` is Coupled or Plugin.

An example of performing a transient simulation for 10 μs 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 157 on page 1125](#).

The parameters `TransientError`, `TransientErrRef`, and `TransientDigits` control the error over the transient integration method. This differs from the error control for the Coupled command, which only controls the error of each nonlinear solution. As with the error parameters for the Coupled command, 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 and Quasistationary command are identical, except that the parameter `t` is identical to the time.

2: Basic Sentaurus Device

Solve Section

In two similar examples, the `Plot` parameter is placed with the other `Transient` parameters, 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 $3.0e-6$.

By placing `Plot` in the `Solve` section (see [Plot, Save, and Load Commands on page 84](#)), this example can be rewritten:

```
Transient( InitialTime = 0.0 FinalTime = 1.0e-5 )
    { Coupled{ Poisson Electron Hole }
        Plot ( Time=( 1.0e-6; 2.0e-6; 3.0e-6 ) NoOverwrite )
    }
```

Extrapolation

If the `Extrapolate` option is present in the `Math` section, the `Transient` command uses the linear extrapolation of the last two solutions to predict the next solution. 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 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 `Transient` options in [Table 5 on page 74](#) control the handling of extrapolation information.

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 [2] 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 [3] 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.

Description of Method

The original method [3] 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 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:

$$\Delta x_n = x_n^F - x_n^I \quad (3)$$

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:

$$x_{n+1}^I = x_n^I - \gamma \frac{x_n^I - x_{n-1}^I}{\Delta x_n - \Delta x_{n-1}} \Delta x_n \quad (4)$$

where γ is a user-defined relaxation factor to stabilize convergence.

2: Basic Sentaurus Device

Solve Section

As Eq. 4 contains uncertainty such as 0/0, especially when Δx 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 Eq. 4 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 Eq. 4 gives $x^I = b$, that is, a desirable cyclic steady-state solution.

It follows that the ratio r :

$$r = -\frac{x_n^I - x_{n-1}^I}{\Delta x_n - \Delta x_{n-1}} \quad (5)$$

can be estimated as $r \approx 1/(1 - \exp(-\alpha t))$. From this, it is clear that because α 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 Eq. 5, Eq. 4 can be rewritten as:

$$x_{n+1}^I = x_n^I + \gamma r \Delta x_n \quad (6)$$

Although Eq. 5 and Eq. 6 are equivalent to Eq. 4, it is more convenient inside Sentaurus Device to use Eq. 5 and Eq. 6. 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 also specify the value of the parameter r_{\min} .

An extrapolation procedure, which is described by Eq. 5 and Eq. 6, 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:

$$\frac{\Delta x}{x + x_{\text{ref}}} < \varepsilon_{\text{cyc}} \quad (7)$$

Values of x_{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_{av}^o of the ratio r is estimated and can be optionally printed. Estimation of r_{av}^o is performed only at such vertices of the object, where the condition:

$$\frac{\Delta x}{x + x_{\text{ref}}} < \frac{\varepsilon_{\text{cyc}}}{f} \quad (8)$$

is fulfilled, that is, the same condition as Eq. 7, but with a possibly different tolerance $\varepsilon_{1_{\text{cyc}}} = \varepsilon_{\text{cyc}}/f$.

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 Eq. 8 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 Eq. 8 is fulfilled, the value of factor r is replaced by the averaged factor r_{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 $r_{\text{min}} = 0$.

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 132 on page 1100](#) 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) )
) { ... }
```

2: Basic Sentaurus Device

Solve Section

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 input file.

Plot, Save, and Load Commands

Specifying input and output through the `File` section has the disadvantage of being predefined for the full simulation. When a simulation is composed of different parts, it is useful to have more control over file input/output. This is performed through the use of the `Plot`, `Save`, and `Load` commands in the `Solve` section.

The `Plot` statement writes files of type `.dat` (DF-ISE format) or `.tdr` (TDR format). The `Save` statement generates files of type `.sav`. Thereby, the `.tdr` and `.sav` files contain additional information:

- Contact biases and currents
- Occupation probability for specified traps
- Ferroelectric history (electric field and polarization) if the ferroelectric model is switched on

The keyword `-Loadable` as an option of a specific `Plot` statement omits the additional information in the `.tdr` file (see [Table 136 on page 1103](#)). This behavior is applied to all `Plot` commands when the keyword `-PlotLoadable` is specified in the global `Math` section (see [Table 149 on page 1112](#)).

The state of a device can be restored from a `.tdr` file (for single-device simulations only) or from a `.sav` file using the `Load` statement. In particular:

- Contact biases specified in the `Electrode` section are overwritten.
- Occupation probability of traps is restored.
- Ferroelectric history is recovered.

NOTE Trap occupation and ferroelectric history are saved or loaded in TDR format only. Interfaces regions are required in the grid file to save and load data on interfaces (see [Interface Plots on page 58](#)).

The `Plot` and `Save` statements can be used at any level of the `Solve` section. The `Load` statement can only be used as a base level of the `Solve` statement. For example, in the case of a transient simulation, `Load` can only be used before or after the transient, not during it. Multiple `Load` and `Save` commands are allowed in one command file.

The commands are defined as:

```
<command> (<parameters-opt>) <system-opt>
```

where `<command>` is either Plot, Save, or Load. Plot refers to data specified in the Plot section. Use the keyword Explicit as an option of a specific Plot statement to write datasets specified in the Plot section only (see [Table 136 on page 1103](#)). The keyword PlotExplicit in the global Math section applies this behavior to all Plot statements (see [Table 149 on page 1112](#)). `<parameters-opt>` is a list of options (see [Table 136 on page 1103](#)).

The plot parameters in [Table 136 on page 1103](#) are also available in the ACCompute option discussed in [ACCoupled: Small-Signal AC Analysis on page 161](#).

Use `<system-opt>` to obtain a list of optional entries delimited by braces (see [Table 128 on page 1094](#)).

If `<parameters-opt>` is omitted, the defaults are used. If no `<system-opt>` is specified, all physical devices and circuits are saved or plotted. The Load statement requires a file prefix. Therefore, `system-opt` must be empty if the current or all mixed-mode device is selected.

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 (see [Table 149 on page 1112](#)).

Example: Solve Section of Input File with Plot, Save, and Load Operations

```
Solve {
    Plugin {
        Poisson
        Plot ( FilePrefix = "output/poisson")
        Coupled { Poisson Electron Hole }
        Plot (FilePrefix = "output/electric" noOverwrite)
    }
    Save
    Coupled { Poisson Electron Hole Temperature }
    Save ( FilePrefix = "output/therm_init_des")
    Transient {
        Coupled { Poisson Electron Hole Temperature }
        Plot ( FilePrefix = "output/trans"
            Time = ( range = (0 1) ;
            range = (0 1) intervals = 4 ; 0.7 )
            NoOverwrite )
    }
}
```

2: Basic Sentaurus Device

Solve Section

```
Load (FilePrefix = "output/therm_init_des")
...
}
```

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.

In the first base-level `Save`, no file prefix is specified. The default file prefix `Save<globalindex>` is used (`Plot<globalindex>` is used for plots). In this example, the simulation writes to the file `save1_des.sav` because it is the second `Save` and the index starts with zero.

The second base-level `Save` writes a file for each physical device, for example, `output/therm_init_des.sav`.

The `Plot` statement in this transient simulation example 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. This format is similar to quasistationaries.

The `Load` statement reads the files `output/therm_init_des.sav` (or the corresponding compressed files), and the simulation continues with the loaded parameters.

Example: Solve Section of Input File with Multiple Save and Load Operations

```
...
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}}
}
...
...
```

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` command. 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 variant:

```
+System ("UNIX command")
```

is used, Sentaurus Device examines the return status. The `System` command is considered to have converged if the return status is zero. Otherwise, it has not converged.

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` statement in the `Solve` section:

```
NewCurrentPrefix = prefix
```

2: Basic Sentaurus Device

Solve Section

This statement appends the given prefix to the default, current file name, and all subsequent Plot statements are directed to the new file. Multiple NewCurrentPrefix statements can appear 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 a NewCurrentPrefix statement.

CurrentPlot Section

The CurrentPlot statement in the Solve section provides full control over the plotting of device currents and circuit currents. By default, currents are plotted 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. If a CurrentPlot statement is present, it determines the exact location of all plot points that are written to the .plt file. Sentaurus Device can still perform computations for some 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 as described in [CurrentPlot Section on page 59](#).

The syntax of the CurrentPlot statement is:

```
CurrentPlot (options) {body}
```

Both `options` and `body` are optional and can be omitted. `options` is a space-separated list, which can consist of the following entries:

`Time = (<float> ; <float> ; <float> ;)`

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:

$$t = a, t = a + \frac{b-a}{n}, \dots, t = b - \frac{b-a}{n}, t = b \quad (9)$$

`Iterations = (<integer>; <integer>; <integer>;)`

The list of integers specifies the iterations for which a plot is required. This option is available for the `Plugin` command.

`IterationStep = <integer>`

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 136 on page 1103](#)).

`Body` is a space-separated list of devices. If `Body` is not present, Sentaurus Device plots all device currents and the circuit (in mixed-mode simulations). If `Body` is present, only the currents of the given devices are plotted. The keyword `Circuit` can be used to request a circuit plot.

2: Basic Sentaurus Device

Solve Section

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, 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 $t = 0$, $t = 10^{-8}$, and $t = 10^{-7}$:

```
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.

Set Command

The `Set` command changes the boundary condition type for an electrode. The `Electrode` section (see [Electrode Section on page 50](#)) specifies the initial boundary condition. 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 `Set (<name> mode current)` or `Set (<name> mode charge)`, respectively.

To change the boundary condition of a charge contact `<name>` to voltage type, use `Set (<name> mode voltage)`. 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. For example, use:

```
Set ("drain" mode current)
```

to change the boundary condition for the voltage contact `drain` from voltage type to current type. If the boundary condition for `drain` was of current type before the `Set` command is executed, nothing happens.

The `Set` command does not change the value of the voltage or current at the electrode. The boundary value for the new boundary condition type results from the solution previously obtained for the bias point at which the `Set` command appears.

An alternative method to change the boundary condition type of a contact is to use the `Quasistationary` statement (see [Quasistationary Command on page 67](#)). This alternative is usually more convenient. However, a goal value for the boundary condition must be specified, whereas the `Set` command 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. In mixed-mode simulations, the `Set` command can be used to determine the boundary conditions at nodes (see [Set, Unset, Initialize, and Hint on page 154](#) and [Set and Unset Section on page 169](#)).

Math Section

The `Math` section is used to specify defaults for the different `Solve` commands. The two types of `Math` entry are device specific and global. Device-specific entries refer to parameters that affect the solve methods of a device. Global entries are device independent and affect the global solution methods. All `Math` keywords are summarized in [Table 149 on page 1112](#).

Device-specific Math Keywords

Device-specific keywords are used in the `Math` section and device-specific `Math` sections. In this section, keywords are grouped by functionality.

Physics-related Math Keywords

The following keywords are strongly related to physical models; however, they are in the `Math` section because they strongly affect the math:

- `ComputeIonizationIntegrals` computes the ionization integrals from the local field maxima. See [Approximate Breakdown Analysis: Poisson Equation Approach on page 333](#).
- `Cylindrical` specifies that the device is simulated using cylindrical coordinates. In this case, a 3D device is specified by a 2D mesh and the vertical axis around which the device is rotated.
- `MetalConductivity` controls the simulation of currents in metals (see [Conductivity of Metals on page 193](#)).
- `NonLocal` controls the generation of nonlocal line meshes (see [Nonlocal Meshes on page 104](#)).
- `RecomputeQFP` recomputes the quasi-Fermi potentials when the electrostatic potential changes.
- `ParallelToInterfaceInBoundaryLayer` 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 interface of the device.

This switch can be useful to avoid nonphysical breakdowns along an interface with a coarse mesh. It may be specified regionwise, in which case it only applies to boundary elements in a given region.

The switch `ParallelToInterfaceInBoundaryLayer` supports two 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 enabled by default. It can be disabled by specifying -ParallelToInterfaceInBoundaryLayer.

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 to switch off mobility and avalanche derivatives. The derivatives are usually computed analytically, but a numeric computation can be used by specifying Numerically.

Discretization Methods

In some very important cases, Sentaurus Device allows the use of different discretization approaches.

The parameter EvEpara controls the discretization of the electric field parallel to the current.

The electrostatic potential can be computed from an arbitrary reference potential level ψ_{ref} . The ConstRefPot parameter allows you to specify Ψ_{ref} explicitly. Otherwise, Sentaurus Device computes ψ_{ref} from the vacuum level, using the following rules:

- If there is silicon in any simulated semiconductor structure, the intrinsic Fermi level of silicon is selected as reference, $\psi_{\text{ref}} = \Phi_{\text{intr}}(\text{Si})$.
- Otherwise, if any simulated device structure contains GaAs, $\psi_{\text{ref}} = \Phi_{\text{intr}}(\text{GaAs})$.
- 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 ψ_{ref} .

The keywords DirectCurrent and CurrentWeighting determine the numeric approach to computing contact currents. The default method 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. CurrentWeighting activates a domain-integration method that uses a solution-dependent weighting function to minimize numeric errors (see [4] for details). With DirectCurrent, the current is computed as the surface integral of the current density over the contact area.

For some models (van Dort quantum correction model and Lombardi mobility model), the electric field normal to the interface (called Enormal) is used. By default, such an interface is the semiconductor-insulator interface.

2: Basic Sentaurus Device

Math Section

To change the default, this interface must be specified explicitly in the `Math` section, using the syntax:

```
Math { ...
    EnormalInterface(regioninterface=["regionK1/regionL1"
        "regionK2/regionL2" ...],
        materialinterface=["materialM1/materialN1" "materialM2/materialN2" ...]
}
```

For the interface definition, Sentaurus Device takes the union of all specified interfaces.

Math Parameters for Transient Analysis

A set of keywords is available in the `Math` section to control transient simulation. Sentaurus Device uses implicit discretization of nonstationary equations and supports two discretization schemes: the trapezoidal rule/backward differentiation formula (TRBDF), which is a default, and the simpler backward Euler (BE) method.

To activate a particular transient method, `Transient=<transient-method>` must be specified, where `<transient-method>` can be TRBDF or BE. Together with error control of nonlinear equations convergence (which is needed for both DC and transient analysis), time-step control is necessary during transient simulation (see [Transient Simulation on page 901](#), where equations and criteria for time-step control are described).

To activate time-step control, `CheckTransientError` must be specified (it is switched off by default, that is, time-step depends only on convergence of Newton iterations). 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. Relative transient error $\varepsilon_{R,tr}$ is defined similarly to ε_R in [Eq. 13](#):

$$\varepsilon_{R,tr} = 10^{-\text{TransientDigits}} \quad (10)$$

Similarly, for $x_{ref,tr}$ and $\varepsilon_{A,tr}$, the equation:

$$x_{ref,tr} = \frac{\varepsilon_{A,tr}}{\varepsilon_{R,tr}} x^* \quad (11)$$

is valid. The keyword `TransientDigits` must be used to specify relative error $\varepsilon_{R,tr}$ in transient simulations. An absolute error in time-step control can be specified by either the keyword `TransientError` ($\varepsilon_{A,tr}$) or `TransientErrRef` ($x_{ref,tr}$), 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 adapts the time step to resolve the evolution of the gate charge properly, even when this charge is small. For small charges, this requires small time steps. However, this is not always needed; for example, when the gate charge switches sign, the modulus of the charge can become arbitrarily small, but the detailed evolution around zero is unimportant. To avoid unnecessary computational expense, EpsCharge specifies a density below which the modulus of the floating-gate charge is considered insignificantly small, and the time evolution is not resolved by adjusting the time step.

NOTE All transient parameters in the Math section, except Transient and EpsCharge, can be overwritten in the Transient command of the Solve section (see [Transient Command on page 79](#)).

Linear Solver-oriented Math Keywords

The Math parameters to the solution algorithms are device independent and must only appear 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 154 on page 1123](#) for available linear solvers). 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 ACSUBMETHOD.

[Table 154 on page 1123](#) lists the options that are available 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.

2: Basic Sentaurus Device

Math Section

All ILS options can be specified within an `ILSrc` statement in the global Math section:

```
Math {
    ILSrc = "
        set (...) {
            iterative (...);
            preconditioning (...);
            ordering (...);
            options (...);
        };
        ...
    "
    ...
}
```

If this statement is present, the external `.ilsrc` file as discussed in the [Solvers User Guide, Chapter 10 on page 67](#) is no longer required.

Additional ILS options can be found in [Table 154 on page 1123](#).

The UMFPACK solver recognizes the parameters shown in [Solvers User Guide, Chapter 4 on page 17](#). These parameters can be specified inside the Math section as follows:

```
method = UMF (PrintLevel = 2
               DenseRow = 0.2
               DenseColumn = 0.2
               AMDdense = 10
               Strategy = 0
               Tolerance_2by2 = 0.01
               Aggressive = 1
               PivotTolerance = 0.2
               SymPivotTolerance = 0.001
               BlockSize = 32
               AllocInit = 0.7
               FrontAllocInit = 0.5
               Scale = 1
               IrStep = 2
               FixQ = 0)
```

The two 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 may produce a small parallel speedup or slightly more accurate results if this option is selected.

Nonlinear Solver-oriented Math Keywords

The Coupled command is sensitive to the Math parameters:

- Iterations
- LineSearchDamping
- NotDamped
- RhsMin
- RhsFactor
- Digits
- Error
- RelErrControl
- ErrRef

Iterations limits the number of Newton iterations, while LineSearchDamping and NotDamped control damping of the Newton iteration. Iterations, LineSearchDamping, and NotDamped in the Math section set the defaults to the equivalent parameters as in the Coupled command (see [Coupled Command on page 65](#)).

RhsMin and RhsFactor add control to the size of the RHS (that is, the residual of the equations). RhsMin sets a maximum RHS value for the convergence to be accepted, and RhsFactor sets a limit to the amount by which the RHS can augment during a single Newton step.

Apart from accepting a solution whenever the RHS falls below RhsMin, during a Solve statement, Sentaurus Device tries to determine the value of an equation variable x , such that the computed update Δx (after k -th nonlinear iteration) is small enough:

$$\frac{\frac{|\Delta x|}{|x^*|}}{\varepsilon_R \left| \frac{x}{x^*} \right| + \varepsilon_A} < 1 \quad (12)$$

where:

$$\varepsilon_R = 10^{-\text{Digits}} \quad (13)$$

and x^* is a scaling constant.

NOTE The condition Eq. 12 only holds for a scalar equation. It can be generalized in the case of a vector of unknowns (see [Fully Coupled Solution on page 904](#)).

2: Basic Sentaurus Device

Math Section

It is clear that [Eq. 12](#) is equivalent to:

$$\frac{|\Delta x|}{|x| + x_{\text{ref}}} < \varepsilon_R \quad (14)$$

where:

$$x_{\text{ref}} = \frac{\varepsilon_A}{\varepsilon_R} x^* \quad (15)$$

By default, Sentaurus Device uses the condition [Eq. 14](#), but the keyword `-RelErrControl` can be used to switch to [Eq. 12](#). For large values of x ($|x| \rightarrow \infty$), the conditions [Eq. 12](#) and [Eq. 14](#) are equivalent to the relative error criterion:

$$\frac{|\Delta x|}{|x|} < \varepsilon_R \quad (16)$$

Conversely, for small values of x ($|x| \rightarrow 0$), the absolute error conditions are:

$$\left| \frac{\Delta x}{x^*} \right| < \varepsilon_A \quad \text{or} \quad |\Delta x| < x_{\text{ref}} \cdot \varepsilon_R \quad (17)$$

respectively. Sentaurus Device uses the expressions [Eq. 12](#) and [Eq. 14](#) to ensure a smooth transition between absolute and relative error control.

If `-RelErrControl` has been specified to disable relative error control, you must specify the value of `Digits` to change the default value of the relative error. Sentaurus Device uses [Eq. 13](#) to define ε_R internally. In absolute error specifications, the values of ε_A and x_{ref} can be defined independently for each equation variable.

NOTE By default, Sentaurus Device uses relative error control. In this case, the (unscaled) absolute error in nonlinear iterations is specified by either the keyword `Error` (ε_A) or `ErrRef` (x_{ref}).

[Table 133 on page 1100](#) lists the default values for the error control criteria.

Carrier concentrations must never be negative. If during a Newton iteration a concentration erroneously becomes negative, 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 cm^{-3}) by `DensLowLimit=<\text{float}>`; the default is $10^{-100} \text{ cm}^{-3}$.

Similarly, lower and upper limits for the lattice temperature and the carrier temperatures exist. The allowed temperature ranges are specified (in K) by `lT_Range=(<\text{float}> <\text{float}>)` (with defaults 50 K and 5000 K) and `cT_Range=(<\text{float}> <\text{float}>)` (with defaults 10 K and 80000 K). These ranges apply both to the temperatures during the Newton iterations and to the final results.

Keywords for Transient and Quasistationary Control

The Quasistationary and Transient commands are sensitive to the Math parameters Extrapolate and BreakAtIonIntegral. The keyword Extrapolate allows the previous solutions of the Quasistationary or Transient to be used to extrapolate a new solution at each step. BreakAtIonIntegral forces a Quasistationary to stop when the largest ionization integral is greater than one.

Break Criteria

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.

Global Contact 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)
    }
    ...
}
```

2: Basic Sentaurus Device

Math Section

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 $\text{A}/\mu\text{m}^2$ is valid for 1D devices; the unit $\text{A}/\mu\text{m}$ is valid for 2D devices; and the unit A is valid for 3D devices.

Global Device Criteria

The device power is equal to $P = \sum I_k \cdot V_k$ 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 $6\times10^{-5}\text{W}/\mu\text{m}$ or the inner power exceeds $2\times10^{-6}\text{W}/\mu\text{m}$.

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^2 , 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 }
    )
    { 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.

2: Basic Sentaurus Device

Math Section

Mixed Mode

All the abovementioned break criteria are also available in mixed mode. In this case, the BreakCriteria section must contains the circuit device name (an exception is the voltage criterion on the circuit node). Examples of the break criteria conditions in mixed mode are:

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

        # device variables
        Voltage(DevName = diode Contact = "anode" MaxValue=10)
        LatticeTemperature(DevName = mos MaxValue = 1000)
        ElectricField(DevName = mos MaxValue=1e6)
        DevicePower(DevName = resistor AbsValue=1e-5)

        ...
    }
    ...
)
```

Parallelization

Sentaurus Device uses thread parallelism to accelerate simulations on shared memory computers. [Table 6](#) gives an overview of the components of Sentaurus Device that have been parallelized.

Table 6 Areas of parallelization

Area	Description
Matrix assembly	Computation of mobility, avalanche, current density, energy flux; assembly of Poisson, continuity, lattice, and temperature equations; modified local-density approximation (MLDA).
Linear solver	Direct solver PARDISO; iterative solver ILS.

The number of threads can be specified in the global Math section of the Sentaurus Device command file:

```
Math {
    Number_of_Threads = number of threads
}
```

You can specify a constant for the number of threads, such as:

```
Number_of_Threads = 2
```

or:

```
Number_of_Threads = maximum
```

where *maximum* is equivalent to the number of processors available on the execution platform.

If desired, the number of threads can also be specified separately for both the assembly and the linear solvers:

```
Math {
    Number_of_Assembly_Threads = number of assembly threads
    Number_of_Solver_Threads = number of solver threads
}
```

Additionally, the values of the following environment variables are checked:

```
SDEVICE_NUMBER_OF_THREADS, SNPS_NUMBER_OF_THREADS
SDEVICE_NUMBER_OF_ASSEMBLY_THREADS
SDEVICE_NUMBER_OF_SOLVER_THREADS
OMP_NUM_THREADS
```

The priority of the various methods for specifying the number of threads is summarized in [Table 7](#).

Table 7 Specification of number of threads

Priority	Matrix assembly	Linear solver
Highest	Number_of_Assembly_Threads	Number_of_Solver_Threads
	Number_of_Threads	Number_of_Threads
	SDEVICE_NUMBER_OF_ASSEMBLY_THREADS	SDEVICE_NUMBER_OF_SOLVER_THREADS
	SDEVICE_NUMBER_OF_THREADS	SDEVICE_NUMBER_OF_THREADS
	SNPS_NUMBER_OF_THREADS	SNPS_NUMBER_OF_THREADS
Lowest		OMP_NUM_THREADS

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
}
```

2: Basic Sentaurus Device

Math Section

Alternatively, the following UNIX environment variables are recognized:

```
SDEVICE_STACKSIZE, SNPS_STACKSIZE
```

By default, Sentaurus Device uses only one thread and the stack size is 1 MB. For most simulations, the default stack size is adequate.

Observe the following recommendations to obtain the best results from a parallel Sentaurus Device run:

- Speedups are only obtained for sufficiently large problems. As a general rule, 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.
- The parallel execution produces different rounding errors. Therefore, the number of Newton iterations may change.

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 Solver on page 253](#)), the nonlocal tunneling model (see [Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 518](#)), and the trap tunneling model (see [Tunneling and Traps on page 357](#)) use nonlocal meshes. The documentation of the former 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

Nonlocal meshes are specified by the keyword `Nonlocal` in the global `Math` section. `Nonlocal` is 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.

For example, with:

```
Math {
    Nonlocal "GateNonLocalMesh" (
        Electrode="Gate"
        Length=5e-7
    )
}
```

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, and:

```
Math {
    Nonlocal "for_tunneling" (
        Barrier(Region="gateoxide")
    )
}
```

constructs a nonlocal mesh named for_tunneling that consists of lines that connect the different sides of the region gateoxide.

For a summary of available options, see [Table 156 on page 1124](#). For a comprehensive description of the construction of a nonlocal mesh, see [Constructing Nonlocal Meshes on page 107](#).

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 106](#).

To visualize nonlocal meshes, use the keyword NonLocal in the Plot section (see [Plot Section on page 58](#)). 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 μm .

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.

2: Basic Sentaurus Device

Math Section

Visualizing Data Defined on Nonlocal Meshes

To visualize data defined on nonlocal meshes:

- In the `File` section (see [File Section on page 49](#)), 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 [Plot Section on page 58](#)). `NonLocalPlot` does not support the `/vector` option. Additionally, the Schrödinger equation provides special-purpose datasets available only for `NonLocalPlot` (see [Visualizing Schrödinger Solutions on page 258](#)).

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 μm . 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.

For example:

```
NonLocalPlot (
    (0 0) (0 1)
) {
    eDensity hDensity
}
```

plots the electron and hole densities for the nonlocal lines close to the coordinates $(0, 0, 0)$ and $(0, 1, 0)$ in the device.

Constructing Nonlocal Meshes

Nonlocal meshes are specified in the global Math section:

```
Math {  
    NonLocal <string> (  
        Barrier(...)  
        RegionInterface=<string>  
        MaterialInterface=<string>  
        Electrode=<string>  
        ...  
    )  
}
```

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:

- By specifying the regions and materials that form the tunneling barrier (keyword `Barrier`).
- By specifying 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 insulator barriers.

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`.

Use `Length=<float>` (in centimeters) to restrict the length on nonlocal lines (to suppress very long tunneling paths).

2: Basic Sentaurus Device

Math Section

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. Sentaurus Device provides no default value for `Length`; all nonlocal meshes must specify `Length` explicitly.

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 1D Schrödinger Equation

For performance reasons, Sentaurus Device solves the 1D Schrödinger equation (see [1D Schrödinger Solver on page 253](#)) 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 Nonlocal Tunneling Model

Sentaurus Device computes the intersections of the nonlocal lines with the boxes (see [Discretization on page 885](#)); to this end, some lines may become longer than `Length`. The nonlocal tunneling model (see [Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 518](#)) 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. 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.

2: Basic Sentaurus Device

Math Section

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. 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, Sentaurus Device also allows you to 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 [Tunneling and Traps on page 357](#)) 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 make adjustments to 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 111](#)). 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 112](#)). 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 to the standard output:

- The largest error according to [Eq. 14, p. 98](#) that occurs anywhere in the device for the equation.
- The vertex where the largest error occurs.
- The coordinates of the vertex.
- The current value of the solution variable for that vertex.

NewtonPlot

Sentaurus Device can plot the solution variable, the errors, the right-hand sides, and the solution updates after each Newton step. To use this feature:

- Use the `NewtonPlot` keyword in the `File` section (see [File Section on page 49](#)) 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.
- Sentaurus Device plots Newton information if `StepSize` drops below a user-defined value. Use `NewtonPlotStep` to specify this limit. `NewtonPlotStep` is a valid option for `Quasistationary`, `Transient`, or `NewtonPlot`. By default, `NewtonPlot` files are saved only when the simulation fails (that is, `StepSize < MinStep`).
- By default, Sentaurus Device writes the current values of the solution variables only. Optionally, specify the data for output as options to `NewtonPlot` in the `Math` section.
- In addition, `MinError` can be specified as an option to `NewtonPlot`. If present, Sentaurus Device writes the file only if the error of the actual iteration is smaller than all previous errors within the current Newton step.

[Table 149 on page 1112](#) lists all of the options to `NewtonPlot`.

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 904](#) 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` command:

```
Coupled (IncompleteNewton) {poisson electron hole}
```

The Jacobian matrix is reused if the following two conditions are satisfied:

$$\|\text{Rhs}_k\| < \text{RhsFactor} \cdot \|\text{Rhs}_{k-1}\| \quad (18)$$

$$\|\text{Update}_k\| < \text{UpdateFactor} \cdot \|\text{Update}_{k-1}\| \quad (19)$$

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 $|\text{Rhs}|$ and $|\text{step}|$.

The parameters RhsFactor and UpdateFactor can be specified as arguments to the IncompleteNewton keyword:

```
IncompleteNewton (UpdateFactor=0.1 RhsFactor=10)
```

The default values are UpdateFactor=0.1 and RhsFactor=10.

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 8](#) lists the available options, the size of a floating-point number, and the precision.

Table 8 Extended precision floating-point arithmetic

Option	Description	Size	Precision
-ExtendedPrecision	double (default)	64 bits	2.22×10^{-16}
ExtendedPrecision	long double	80 bits (AMD, Linux)	1.08×10^{-19}
		128 bits (IBM)	2.47×10^{-32}
		128 bits (Sun)	1.93×10^{-34}
ExtendedPrecision(128)	double-double	128 bits	4.93×10^{-32}
ExtendedPrecision(256)	quad-double	256 bits	1.22×10^{-63}

On AMD and Linux, 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. On IBM and Sun, the long double data type is implemented in software, and the run-time performance slows down by a factor of 10.

2: Basic Sentaurus Device

Math Section

The data types 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 run-times increase by a factor of 10 and 100, respectively.

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 may or may not be beneficial for general convergence problems.

When using extended precision, the following points should be observed for the best results:

- Use the linear solver SUPER. It is the only linear solver that supports extended precision. The second best choice is PARDISO with the options RecomputeNonSymmetricPermutation and IterativeRefinement.
- In the Math section:
 - Increase the value of Digits. Possible values are Digits=15 for ExtendedPrecision(128) and Digits=25 for ExtendedPrecision(256).
 - Decrease the value of RhsMin. Possible values are RhsMin=1e-15 for ExtendedPrecision(128) and RhsMin=1e-25 for ExtendedPrecision(256).
 - Slightly increase Iterations, for example, from 15 to 20.

Some features of Sentaurus Device do not take advantage of extended precision. They include:

- Linear solvers. The exception is SUPER, which supports all extended precision formats.
- Input and output (DF–ISE and TDR file formats).
- Compact circuit models (refer to the *Compact Models User Guide*).
- CMI and PMI (see Part V of this manual).
- Monte Carlo (refer to the *Sentaurus Device Monte Carlo User Guide* for more information).
- Optical and laser simulations (see Part III of this manual).
- Integration of beam distribution for optical generation (see the keyword RecBoxIntegr in [Optical Beam Absorption on page 434](#)).

Specifying Model Parameters

Specifying Region and Material Parameters

It is possible to redefine parameters for some or all physical models and to restrict the redefined parameters to certain regions and materials. For the simulation of new materials for which physical parameters are not well established, it is important to be able to modify the default parameters. These redefinitions are performed by changing the parameter file as described in the following sections.

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 9](#) lists the principal options for the command `sdevice -P`.

Table 9 Principal options for generating 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> . The first section of the parameter file does not contain material reference. It includes all models and parameters available inside Sentaurus Device. Sections with references to specific materials include only the default models defined for the particular material.
<code>-P:Material</code>	Prints model parameters for the specified material.
<code>-P:Material:x</code>	Prints model parameters for the specified material for mole fraction x .
<code>-P:Material:x:y</code>	Prints model parameters for the specified material for mole fractions x and y .
<code>-P filename</code>	Prints model parameters for materials and interfaces used in the command file <code>filename</code> .
<code>-r</code>	Reads parameters from material library before printing them (see Library of Materials on page 117).

NOTE One restriction to the use of a parameter file is that only one file is permitted in any simulation. Therefore, the parameter sections for all the various materials and regions must be concatenated into a single .par file, which can be unwieldy and lead to errors. It is more convenient to create a ‘library’ of parameter files (see [Library of Materials on page 117](#)).

Changing Parameter Values in Parameter File

The section of the parameter file for predefined materials is:

```
Material = "material" {  
    <parameter file body>  
}
```

For example, the beginning of the silicon parameter file is:

```
Material = "Silicon" {  
    Epsilon  
    { * Ratio of the permittivities of material and vacuum  
        epsilon = 11.7    # [1]  
    }  
    ...  
}
```

For region-specific parameter specifications, the syntax is:

```
Region = "region-name" {  
    <parameter file body>  
}
```

In the region and material parameter specifications, any model and parameter from the default section is usable, even if it is not printed for the particular material. Some models offer a choice between different representations or formulas for the same (or equivalent) physical parameters, for example, effective mass or density-of-states (DOS). To choose a particular representation, the parameter `Formula` is usually selected.

Library of Materials

As a more flexible alternative to the use of a parameter file, Sentaurus Device supports a library of material parameters. A library is a collection of parameter files for individual materials, interfaces, and electrodes contained in a specific directory. By default, Sentaurus Device assumes a location of the library in the directory:

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

If a user-specific library is required or a group of users need to create a shared parameter library, it is necessary to create the library as a directory and define a new environment variable SDEVICEDB with a path to that directory. The library must contain a number of parameter files with each material having a separate parameter file, for example, Silicon.par, GaAs.par, and Oxide.par.

The command option `sdevice -L` is used to create such files. [Table 10](#) lists the options of this command.

Table 10 Principal options for creating a library of materials

Option	Description
<code>-L:All</code>	Creates separate parameter files for all materials that Sentaurus Device recognizes.
<code>-L:Material</code>	Creates a model parameter file for a specified material.
<code>-L:Material:x</code>	Creates a model parameter file for a specified material for mole fraction x.
<code>-L:Material:x:y</code>	Creates a model parameter file for a specified material for mole fractions x and y.
<code>-L <inputfile.cmd></code>	Sentaurus Device automatically examines the device structure to be simulated. It scans the .tdr file specified in the File section of the input file <inputfile.cmd>. Separate parameter files for each material, interface, and electrode found in the .tdr file are created, and a parameter file models.par is written.
<code>-r</code>	Reads parameters from material library before printing them.

The command file pp1_des.cmd of Sentaurus Device contains the following File section:

```
File {* input files:  
    Grid = "nmos_mdr.tdr"  
    ...  
}
```

2: Basic Sentaurus Device

Specifying Model Parameters

By using the command:

```
sdevice -L pp1_des.cmd
```

a parameter file is created in the current directory for each material, material interface, and electrode found in nmos_mdr.tdr. In this example, the appropriate files are Silicon.par, Oxide.par, Oxide%Silicon.par, Oxide%Oxide.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" }
Region = "Region3" { Insert = "Oxide.par" }
Region = "Region4" { Insert = "Oxide.par" }
Region = "Region5" { Insert = "Oxide.par" }
RegionInterface = "Region1/Region0" { Insert = "Oxide%Silicon.par" }
RegionInterface = "Region4/Region1" { Insert = "Oxide%Oxide.par" }
RegionInterface = "Region3/Region4" { Insert = "Oxide%Oxide.par" }
RegionInterface = "Region5/Region3" { Insert = "Oxide%Oxide.par" }
RegionInterface = "Region5/Region1" { Insert = "Oxide%Oxide.par" }
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, but 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"  
...  
}
```

NOTE The models.par file contains a list of references to other parameter files by using the keyword Insert. Generally, any parameter file can have inserts of other parameter files; the files can be nested. An inserted file defines defaults, and it is easy to change default values after an Insert statement as required. Such a parameter file is easy to read and minimizes mistakes in parameter definition.

Sentaurus Device uses the following strategy to search for inserted files. First, the current simulation directory is checked. If the file does not exist, definition of the environment variable SDEVICE is verified. If SDEVICE is defined, Sentaurus Device checks the directory associated with the variable. Otherwise, Sentaurus Device checks the default library location \$STROOT/tcad/\$STRELEASE/lib/sdevice/MaterialDB. In all cases, Sentaurus Device shows a real path to the file and displays an error message if it cannot be found.

NOTE The environment variable `SDEVICEDEB` is also used to locate the file `Molefraction.txt` (see [Library of Materials on page 117](#)).

NOTE The insert directive is also available for command files (see [Inserting Files on page 48](#)).

Parameters of Compound Materials

A detailed parameter representation of compound materials in Sentaurus Device is described in [Ternary Semiconductor Composition on page 552](#). The command to print default parameters of such materials is the same as for regular monomaterials (see [Generating a Copy of Parameter File on page 115](#)), but for certain models (see [Composition-dependent Models on page 551](#)), it prints polynomial coefficients of approximations, which are dependent on a composition fraction.

Sometimes, it is difficult to analyze such coefficients to obtain a real value of physical models (for example, the band gap) for certain composition mole fractions. By using the command `sdevice -M <inputfile.cmd>`, Sentaurus Device creates a `models-M.par` file, which 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 default material parameters.

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. It is useful for noncritical models, but it can lead to confusion, for example, if the semiconductor band gap is not defined, Sentaurus Device uses a silicon one. 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.

2: Basic Sentaurus Device

Specifying Model Parameters

Table 11 lists the critical models (with names from the parameter file of Sentaurus Device) with materials where these models are checked.

Table 11 List of critical models

Model	Insulator	Semiconductor	Conductor
Auger		x	
Bandgap		x	
ConstantMobility		x	
DopingDependence		x	
eDOSmass hDOSmass		x	
Epsilon	x	x	
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 keyword
-CheckUndefinedModels in the `Math` section.

Default Parameters

Sentaurus Device provides built-in default parameters for many models and materials. These parameters can be viewed using the `sdevice -L` command (see [Library of Materials on page 117](#)).

However, Sentaurus Device also offers the option to read default parameters from files. This option is activated by the keyword `DefaultParametersFromFile` in the global `Physics` section:

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

Table 12 lists the file names that are used to initialize default parameters.

Table 12 Filenames for default parameters

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

Sentaurus Device uses the same search strategy as for inserted files (see [Library of Materials on page 117](#)). If a matching file is not found, the built-in default parameters are used. Check the log file of Sentaurus Device to see which files were actually used.

Hierarchy of Models and Parameters

Both models (specifications in the command file) and parameters (specifications in the parameter file) can be selected for different locations: globally, regionwise, materialwise, or interface-wise. This section discusses how specifications for different locations are related.

Model Hierarchy

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 region-specific and material-specific Physics sections overlap, the region-specific declaration overrides the material-specific declaration.

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, <GaAs models> is not added.

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

Parameter Hierarchy

The methodology used for the specification of a parameter is similar to the hierarchy of models described in [Model Hierarchy on page 121](#):

- Default parameters are defined in the parameter file section for the particular material.
- NOTE** The parameters of a particular model may not be defined for a given material. However, if this model is used and parameters are not redefined in the appropriate region or material section of the parameter file, parameters of the default material (silicon) are used.
- The material section of the parameter file allows you to overwrite default values for a particular material.
 - The region section also overwrites the default values for the appropriate material. However, if for the region `reg` with material `mat`, both the region and material sections of the parameter file are defined, changes from the `mat` section are valid in all regions with this material except the region `reg`. In the region `reg` only, the changes from the region sections of the parameter file are applied.

Material and Doping Specification

Sentaurus Device supports all materials that are declared in the file `datexcodes.txt` (see [Utilities User Guide, Chapter 2 on page 27](#)).

By default, Sentaurus Device supports all typical species of silicon technology (donors: As, P, and Sb, and acceptors: B and In) and allows user-defined ones (see [User-Defined Species on page 125](#)). Usually, nitrogen is not recognized as a doping species. To enable nitrogen as a donor, use the keyword `UseNitrogenAsDopant` in the global `Physics` section.

Sentaurus Device loads doping distributions from the input doping file specified in the `File` section (see [File Section on page 49](#)) and reads the following datasets:

- Net doping (`DopingConcentration` in the input doping file)
- Total doping (`TotalConcentration` in the input doping 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.
- The same rule is applied for the total doping dataset.

NOTE Total concentration, which originates from process simulators (for example, 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 input doping file (for example, `BoronActiveConcentration`). Otherwise, the chemical dopant concentration is used (for example, `BoronConcentration`).

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

- The net doping concentration, $N_{\text{net}} = N_{\text{D},0} - N_{\text{A},0}$.
- $N_{\text{D},0}$ and $N_{\text{A},0}$, the donor and acceptor concentrations.
- The total doping concentration $N_{\text{D},0} + N_{\text{A},0}$.

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

1. If the input doping file does not have any species: N_{net} is initialized from `DopingConcentration`, N_{tot} is initialized from `TotalConcentration` or $|N_{\text{net}}|$ if `TotalConcentration` was not read, $N_{\text{D},0} = (N_{\text{tot}} + N_{\text{net}})/2$, and $N_{\text{A},0} = (N_{\text{tot}} - N_{\text{net}})/2$.

2. If the input doping file has individual species: $N_{A,0}$ and $N_{D,0}$ are computed as the sum of individual donor and acceptor concentrations, N_{net} is initialized from DopingConcentration or $N_{D,0} - N_{A,0}$ if DopingConcentration was not read, and N_{tot} is initialized from TotalConcentration or $N_{A,0} + N_{D,0}$ if TotalConcentration was not read.
3. If the command file contains trap levels (see [Chapter 10 on page 349](#)) in the Physics section with the keyword Add2TotalDoping (see [Table 224 on page 1176](#)), they are added to the respective doping concentrations.

User-Defined Materials

Sentaurus Device can manage arbitrary materials in devices, and new materials can be defined in the local working directory. The following search strategy is observed to locate the datexcodes.txt files:

- `$STROOT_LIB/datexcodes.txt` or `$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 earlier files. In this way, the local file only needs to contain new materials that you want to add.

NOTE The DATEX environment variable is no longer used.

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 such as Tecplot SV.

The group value identifies the type of new material. The available values are:

- Conductor
- Insulator
- Semiconductor

The field `color` defines the color of the material in visualization tools. This field must have the syntax:

```
color = #rrggbb
```

where `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 13](#) lists sample values for color.

Table 13 Sample color values

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

[Mole-Fraction Materials on page 548](#) discusses how compound semiconductors can be defined in Sentaurus Device.

User-Defined Species

Sentaurus Device supports the most important dopants used in silicon technology: the donors As, P, Sb, N, and the acceptors B and In. For the simulation of other semiconductors such as III-V compounds, the actual dopants (such as Si and Be) are supported only through user-defined species. All such species used during a simulation must be declared in the Variables section of the file `datexcodes.txt` (see [Material and Doping Specification on page 123](#)). If the incomplete ionization model is activated (see [Chapter 6 on page 245](#)), the model parameters of user-defined species must be specified in the Ionization section of the material parameter file.

2: Basic Sentaurus Device

Material and Doping Specification

The following example shows the definition of two species for SiC material, with N as a donor and Al as an acceptor. The file `datexcodes.txt` is:

```
Variables {
    ...
    Nitrogen {
        code = 960
        label = "total (chemical) Nitrogen concentration"
        symbol = "NitrogenTotal"
        unit = "/cm3"
        factor = 1.0e+12
        precision = 4
        interpol = log
        material = Semiconductor
        doping = donor(ionized_code = 961)
    }
    NitrogenPlus {
        code = 961
        label = "Nitrogen+ concentration (incomplete ionization)"
        symbol = "N-Nitrogen+"
        unit = "/cm3"
        factor = 1.0e+12
        precision = 4
        interpol = log
        material = Semiconductor
    }
    Al {
        code = 962
        label = "total (chemical) Al concentration"
        symbol = "AlTotal"
        unit = "/cm3"
        factor = 1.0e+12
        precision = 4
        interpol = log
        material = Semiconductor
        doping = acceptor(ionized_code = 963)
    }
    AlMinus {
        code = 963
        label = "Al- concentration (incomplete ionization)"
        symbol = "P-Al-"
        unit = "/cm3"
        factor = 1.0e+12
        precision = 4
        interpol = log
        material = Semiconductor
    }
    ...
}
```

Therefore, each new dopant must have two subsections in the file `datexcodes.txt`. One subsection must have the field:

```
doping = type(ionized_code = <code value>)
```

where `type` can be a donor or an acceptor, and `ionized_code` is the code of another subsection that corresponds to the ionized dopant concentration.

User-defined species can be plotted by specifying the following keywords in the `Plot` section:

```
Plot {  
    UserSpeciesConcentration  
    UserSpeciesIncompleteConcentration  
}
```

If the option `UserSpeciesIncompleteConcentration` or `UserSpeciesConcentration` is activated, all user-defined species are saved.

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 commands are available (refer to the *Inspect User Guide* for more information). These commands are useful for the purpose of parameter extraction (see [Extraction on page 132](#)). However, the graphical functionality of the Inspect commands has been disabled.

Overview

An entire device simulation can be performed by 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{
```

2: Basic Sentaurus Device

Tcl Command File

```
{ 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{
        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_inital_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_inital_save\)

        QuasiStationary (
            InitialStep=0.25 Increment=1.3
            MaxStep=0.25 MinStep=1e-4
            Goal {Name=\gate\ Voltage=$VT}
        )
        { Coupled {Poisson Electron Hole} }
    }
"
"
```

2: Basic Sentaurus Device

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.

sdevice Command

The Tcl `sdevice` command expects a single argument. This argument 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

The Tcl `sdevice_init` command expects a single argument. This argument 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 abort.

sdevice_solve Command

The Tcl `sdevice_solve` command expects a single argument. It can only be used 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

The Tcl `sdevice_finish` command expects no arguments, and it 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

The Tcl `sdevice_parameters` 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  
    }  
}
```

Flowchart

Figure 20 shows the sequence in which the various Tcl `sdevice` commands can be invoked.

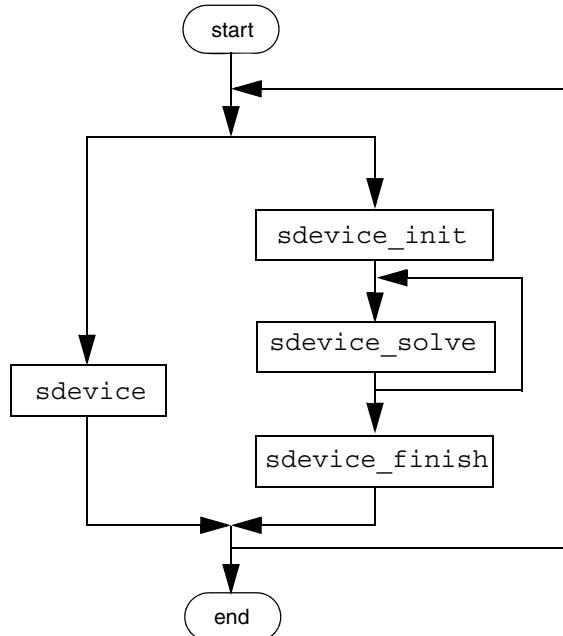


Figure 20 Flowchart of sequence for Tcl commands

2: Basic Sentaurus Device

Tcl Command File

Extraction

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"
        Plot     = "extract_VT_des.tdr"
        Current  = "extract_VT_des.plt"
        Output   = "extract_VT_des.log"
    }

    Physics { ... }

    Solve {
        QuasiStationary (
            InitialStep=0.05 MaxStep=0.05
            Goal {Name="gate" Voltage=2}
        )
        { Coupled {Poisson Electron Hole} }
    }
}

proj_load extract_VT_des.plt proj
cv_create idvgs "proj gate OuterVoltage" "proj drain TotalCurrent"
set VT [f_VT idvgs]
puts stdout "threshold voltage VT = ${VT}V"
```

List of Available Inspect 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
```

Two-port network RF extraction:

```
tpnx_ac2h,          tpxn_ac2s,        tpxn_ac2y,        tpxn_ac2z,
tpnx_characteristic_impedance, tpxn_DevWidth,   tpxn_GetBias,
tpnx_GetParsAtPoint,   tpxn_list_Bias,   tpxn_list_fq,   tpxn_load,
tpnx_setdBPoint, tpxn_startfq, tpxn_y2MSG, tpxn_y2MUG
```

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
```

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
```

2: Basic Sentaurus Device

Saving Snapshots

Output Redirection

The Tcl commands `sdevice`, `sdevice_init`, `sdevice_solve`, and `sdevice_finish` allow you to redirect standard output and standard errors. The syntax is shown in [Table 14](#).

Table 14 Output redirection

Syntax	Description
<code>> filename</code>	Writes standard output to <code>filename</code> .
<code>2> filename</code>	Writes standard error to <code>filename</code> .
<code>>& filename</code>	Writes both standard output and standard error to <code>filename</code> .
<code>>> filename</code>	Appends standard output to <code>filename</code> .
<code>2>> filename</code>	Appends standard error to <code>filename</code> .
<code>>>& filename</code>	Appends both standard output and standard error to <code>filename</code> .

Known Restrictions

The following solve commands only apply to the subsequent statements within a single `Solve` section. They do not apply beyond an `sdevice_solve` command:

- `NewCurrentPrefix` (see [NewCurrentPrefix Statement on page 87](#))
- `Set (TrapFilling=...)` (see [Explicit Trap Occupation on page 361](#))

If a transient simulation is performed using several `sdevice_solve` commands, the correct initial time must be specified. Sentaurus Device does not remember the last transient time from the previous `sdevice_solve` command.

Saving 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.

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 behavior changes if `Interrupt=BreakRequest` or `Interrupt=PlotRequest` is specified in the Math section (see [Table 149 on page 1112](#)). 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 abort 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.

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:

1. A header identifying the file format.
2. A section containing the process information of the MOSFET such as channel length or channel width.
3. A collection of curves containing 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
```

2: Basic Sentaurus Device

Extraction File

```
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
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

[Table 15](#) shows the supported analysis modes.

Table 15 Analysis modes

Analysis	Curve	Description
DC	I _i _V _j	The i -th terminal current versus j -th terminal voltage. $i,j=b$ (bulk), d (drain), g (gate), or s (source) Bias conditions: all terminal voltages other than j -th terminal voltage.
AC	A _i _j_V _k	Admittance A_{ij} versus k -th terminal voltage. $i,j,k=b$ (bulk), d (drain), g (gate), or s (source) Bias conditions: all terminal voltages other than k -th terminal voltage and frequency.
	A _i _j_F	Admittance A_{ij} versus frequency. $i,j=b$ (bulk), d (drain), g (gate), or s (source) Bias conditions: V_b, V_d, V_g, V_s .
	C _i _j_V _k	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.
	C _i _j_F	Capacitance C_{ij} versus frequency. $i,j=b$ (bulk), d (drain), g (gate), or s (source) Bias conditions: V_b, V_d, V_g, V_s .
Noise	noise_V _i _V _j	Autocorrelation noise voltage spectral density (NVSD) for electrode i versus j -th terminal voltage. $i,j=b$ (bulk), d (drain), g (gate), or s (source) Bias conditions: all terminal voltages other than j -th terminal voltage and frequency.
	noise_V _i _F	Autocorrelation noise voltage spectral density (NVSD) for electrode i versus frequency. $i=b$ (bulk), d (drain), g (gate), or s (source) Bias conditions: V_b, V_d, V_g, V_s .
	noise_I _i _V _j	Autocorrelation noise current spectral density (NISD) for electrode i versus j -th terminal voltage. $i,j=b$ (bulk), d (drain), g (gate), or s (source) Bias conditions: all terminal voltages other than j -th terminal voltage and frequency.
	noise_I _i _F	Autocorrelation noise current spectral density (NISD) for electrode i versus frequency. $i=b$ (bulk), d (drain), g (gate), or s (source) Bias conditions: V_b, V_d, V_g, V_s .

2: Basic Sentaurus Device

Extraction File

File Section

The name of the extraction file can be specified in the `File` section of the Sentaurus Device command file:

```
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). For other contact names, it is possible to 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/doping file. It is also possible to specify the same information in an `Extraction` section:

```
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/doping file.

Solve Section

The Quasistationary command in the Solve section supports an Extraction option to request voltage-dependent extraction curves. Multiple curves can be generated during a single ramp. No extraction curves are produced if the Extraction option is missing:

```

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 } }
}

```

2: Basic Sentaurus Device

References

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 15 on page 137](#).

References

- [1] 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.
- [2] K. S. Kundert, J. K. White, and A. Sangiovanni-Vincentelli, *Steady-State Methods for Simulating Analog and Microwave Circuits*, Boston: Kluwer Academic Publishers, 1990.
- [3] 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.
- [4] 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.

This chapter describes the commands needed for mixed-mode simulations.

Overview

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 21](#)), and solve commands must be specified that solve the whole system of devices.

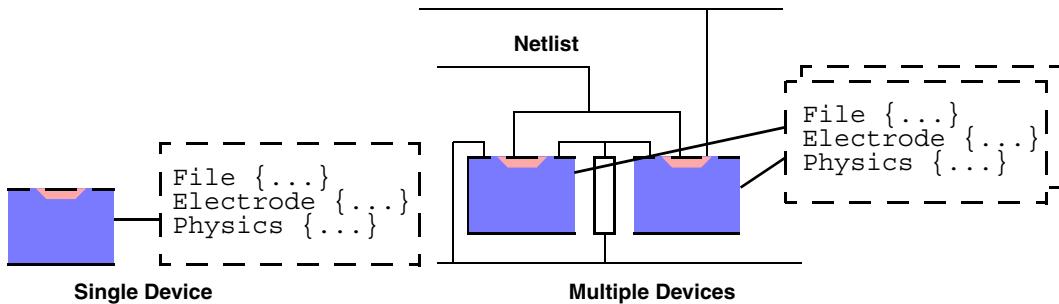


Figure 21 Each device in a multidevice simulation is connected with a circuit netlist

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.

Sentaurus Device also provides a number of compact models for use in mixed-mode simulations.

Compact Models

Sentaurus Device provides four different types of model:

- SPICE

These include compact models from Berkeley SPICE 3 Version 3F5. The BSIM3v3.2, BSIM4.1.0, and BSIMPDv2.2.2 MOS models are also available.

- HSPICE

Several frequently used HSPICE models are provided.

- Built-in

There are several special-purpose models.

- User-defined

A compact model interface (CMI) is available for user-defined models. These models are implemented in C++ and linked to Sentaurus Device at run-time. No access to the source code of Sentaurus Device is necessary.

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

Hierarchical Description of Compact Models

In Sentaurus Device, the compact models comprise three levels:

- Device

This describes the basic properties of a compact model and includes the names of the model, electrodes, thermodes, and internal variables; and the names and types of the internal states and parameters. The 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, a parameter set defines parameters that are shared among several instances. Most SPICE 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. If necessary, an instance can override the values of its parameters.

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

The parameter sets and instances in a circuit simulation are specified in external SPICE circuit files (see [SPICE Circuit Files on page 147](#)). These files are recognized by the extension .scf and are parsed by Sentaurus Device at the beginning of a simulation.

Table 16 lists the built-in models and **Table 17** presents an overview of the SPICE models.

Table 16 Built-in models in Sentaurus Device

Model	Device	Default parameter set
Electrothermal resistor	Ter	Ter_pset
Parameter interface	Param_Interface_Device	Param_Interface
SPICE temperature interface	Spice_Temperature_Interface_Device	Spice_Temperature_Interface

Table 17 SPICE models in Sentaurus Device

Model	Device	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
Junction diode	Diode	Diode_pset
Bipolar junction transistor	BJT	BJT_pset
Junction field effect transistor	JFET	JFET_pset

3: Mixed-Mode Sentaurus Device

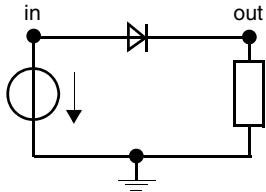
Overview

Table 17 SPICE models in Sentaurus Device

Model	Device	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
GaAs MESFET	MES	MES_pset
HSPICE Level 1	HMOS_L1	
HSPICE Level 2	HMOS_L2	
HSPICE Level 3	HMOS_L3	
HSPICE Level 28	HMOS_L28	
HSPICE Level 49	HMOS_L49	
HSPICE Level 53	HMOS_L53	
HSPICE Level 54	HMOS_L54	
HSPICE Level 57	HMOS_L57	
HSPICE Level 59	HMOS_L59	

Example: Compact Models

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.

3: Mixed-Mode Sentaurus Device

Overview

NOTE A parameter set must be declared before it can be referenced by an instance.

The *Compact Models User Guide* further explains the SPICE parameters in this example. The input 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 `.scf` files (SPICE circuit files).

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 plot file `rectifier_des.plt`. The simulation time, the voltages of the nodes `in` and `out`, and the current from the resistor `r` into the node `0` are plotted. The `Solve` section describes the simulation. The keyword `Circuit` denotes the circuit equations to be solved.

The instances in a circuit can also appear directly in the `System` section of the input file, for example:

```
System {
    Vsource_pset v (in 0) {sine = (0 5 1meg 0 0)}
    D1n4148 d1 (in out) {temp = 30}
    Resistor_pset r (out 0) {resistance = 1000}

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

SPICE Circuit Files

Compact models can be specified in an external SPICE circuit file, which is recognized by the 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 18](#) lists the possible parameter types.

Table 18 Parameters in SPICE circuit files

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

Similarly, the 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, ...`.

3: Mixed-Mode Sentaurus Device

Device Section

This instance also defines or overrides parameter values. The complete syntax of the input language for SPICE circuit files is given in [Compact Models User Guide, Syntax of Compact Circuit Files on page 137](#).

NOTE The tool `spice2sdevice` is available to convert Berkeley SPICE circuit files (extension `.cir`) to Sentaurus Device circuit files (extension `.scf`) (see [Utilities User Guide, Chapter 7 on page 73](#)).

Device Section

The `Device` sections of the input 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 the `Electrode`, `Thermode`, `File`, `Plot`, `Physics`, and `Math` sections. For example:

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

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

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

System Section

The System section defines the netlist of physical devices and circuit elements to be solved. The netlist is connected through circuit nodes. 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. The node 0 is predefined as the ground node (0 V).

Compact models can be defined in SPICE circuit files (see [SPICE Circuit Files on page 147](#)). However, instances of compact models can also appear directly in the System section of the input file:

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

The order of the nodes in the connectivity list corresponds to the electrodes and thermodes in the SPICE device definition (refer to the *Compact Models User Guide*).

The connectivity list is a list of contact-name=node-name connections, separated by white space. Contact-name is the name of the contact from the grid file of the given device, and 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 node. For example, the following defines a physical diode and 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 the electrodes (see [Compact Models User Guide, Syntax of Compact Circuit Files on page 137](#)). 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 input file (with Device sections) or an external .device file.

3: Mixed-Mode Sentaurus Device

System Section

The System section can contain the initial conditions of the nodes. Three types of initialization can be specified:

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

In addition, the Unset command is available to free a node after a Set command.

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 the 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 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 enables electrodes and thermodes to share the same contact name or number.

Circuit Devices

SPICE instances can be declared in SPICE circuit files as discussed in [SPICE Circuit Files on page 147](#). They can also appear directly in the System section of the input file, for example:

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

This declaration in the input file provides the same information as the equivalent declaration in the SPICE circuit file.

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 (refer to the *Compact Models User Guide* for a list of the internal nodes for each model).

Electrical and Thermal Netlist

Sentaurus Device allows both electrical and thermal netlists to 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))  
}
```

3: Mixed-Mode Sentaurus Device

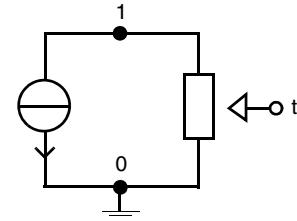
System Section

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 t_{300} through two thermal resistors r_a and r_c .

The Plot statement accesses the voltage of the node a , the temperature of the nodes t_a and t_c , the heat flux from pres into t_a and t_c , 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 :



$$r_0 = \text{rsh} \cdot \frac{l - \text{narrow}}{w - \text{narrow}} \quad (20)$$

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

To feed the value of the thermal node t as a parameter into the 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 the 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 is used to initialize 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 so-called 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:

$$\text{Spice temperature} = \text{offset} + c_1 u + c_2 u^2 + c_3 u^3 \quad (21)$$

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 .

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

```

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}
    }
}
```

Set, Unset, Initialize, and Hint

The keywords Set, Initialize, and Hint are used to set nodes to a specific value.

Set establishes the node value. This value remains fixed during all subsequent simulations until a Set or an Unset command is used in the Solve section (see [Set and Unset Section on page 169](#)), or the node becomes a Goal of a Quasistationary, which controls the node itself (see [Quasistationary Command on page 160](#)). 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 commands exist in the Solve section. These act like the System-level Set but allow more flexibility (see [System Section on page 149](#)).

The Set command can also be used to change the value of a parameter in a compact model. For example, the resistance of the resistor r1 changes to 1000Ω :

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

Initialize 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 keyword Initialize can be used with an internal node to set a current through an inductor before a transient simulation. For example, the keyword:

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

Hint 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 commands are used as follows:

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

For example:

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

System Plot

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 sent to a given file name. If no file name is provided, the output is sent to the standard output file and the log file. The Plot statement is:

```
Plot (<plot command list>
      Plot <filename> (<plot command list>)
```

where <plot command list> is a list of nodes or plot commands as defined in [Table 145 on page 1109](#).

NOTE Plot commands are case sensitive.

Two examples are:

```
Plot (a b i(r1 a) p(r1 rT))
      Plot "plotfile" (time() v(a b) i(d1 a))
```

NOTE The Plot command does not print the time by default. When plotting a transient simulation, the time() command must be added.

AC System Plot

An ACPlot statement in the System section can be used 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 system plot command discussed in [System Plot on page 155](#).

If an ACPlot statement is present, 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.

File Section

In the `File` section, one of the following can be specified:

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

The syntax for these keywords is shown in [Table 127 on page 1091](#).

The variables `Output` and `ACExtract` can be assigned a file name. Only a file name without an extension is required. Sentaurus Device automatically appends a predefined extension:

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

The variables `DevicePath`, `SPICEPath`, and `CMIPath` represent search paths. They must be assigned 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. They are not overwritten by a definition with the same name in the command file.

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

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

Device-specific keywords are defined under the file entry in the `Device` section (see [File Section on page 49](#)).

SPICE Circuit Models

Sentaurus Device supports SPICE circuit models for mixed-mode simulations. These models are based on Berkeley SPICE 3 Version 3F5. Several frequently used HSPICE models are also available. A detailed description of the SPICE models can be found in the *Compact Models User Guide*.

User-Defined Circuit Models

Sentaurus Device provides a compact model interface (CMI) for user-defined circuit models. The models are implemented in C++ by you and linked to Sentaurus Device at run-time. 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. This compact circuit file is read by Sentaurus Device before the model is loaded.
3. Implement a set of interface subroutines C++. Sentaurus Device provides a run-time environment.
4. Compile the model code into a shared object file, which is linked at run-time to Sentaurus Device. A `cmi` script executes this compilation.
5. Use the variable `CMIPath` in the `File` section of the command file to define a search path.
6. Reference user-defined compact models in compact circuit files (with the extension `.ccf`) or directly in the `System` section of the input file.

3: Mixed-Mode Sentaurus Device

Solve Section

Solve Section

The `Solve` commands do not fundamentally change when used in mixed mode. The major differences are with the `Coupled` command, the extra goals available for the `Quasistationary` command, and the `ACCoupled` and `HBCoupled` commands that only work in mixed mode, and the `Continuation` command that only works in single-device mode.

Coupled Command

The `Coupled` command in mixed mode addresses contact and circuit equation variables. Devices can be selected individually or together.

Circuit and Contact Equation–Variable Keywords

The `Coupled` command takes a set of equation–variable pairs as parameters. The `Contact` and `Circuit`, and `TContact` and `TCircuit` equation–variable pairs are introduced for mixed-mode problems. [Table 19](#) provides the full list of equation–variable pair keywords. These four keywords are accessible if the keyword `NoAutomaticCircuitContact` is specified in the `Math` section (at the top level).

Table 19 Keywords of equation-variable pairs for Coupled command

Keyword	Corresponding equation	Corresponding variable
Poisson	Poisson	Electrostatic potential
Contact	Electrical contact interface	Electrostatic potential
TContact	Thermal contact interface	Temperature
Circuit	Electrical circuit	Voltage, current
TCircuit	Thermal circuit	Temperature
Electron	Electron continuity	Electron density
Hole	Hole continuity	Hole density
Temperature	Temperature	Temperature
eTemperature	Electron temperature	Electron temperature
hTemperature	Hole temperature	Hole temperature

The keyword `Circuit` controls the resolution of the electrical circuit models and nodes. `Contact` controls the resolution of the electrical interface condition at the contacts. Similarly,

`TContact` and `TCircuit` control the resolution of the thermal interface condition and thermal circuit, respectively. By default, the keywords `TContact` and `TCircuit` are not used because the Poisson equation-variable also covers the contact and circuit domains.

When the keyword `NoAutomaticCircuitContact` is used in the `Math` statement, the `Poisson` keyword only covers the device. The `TCircuit` and `TContact` keywords can then be used for the circuit and contact parts (see [Figure 22](#)).

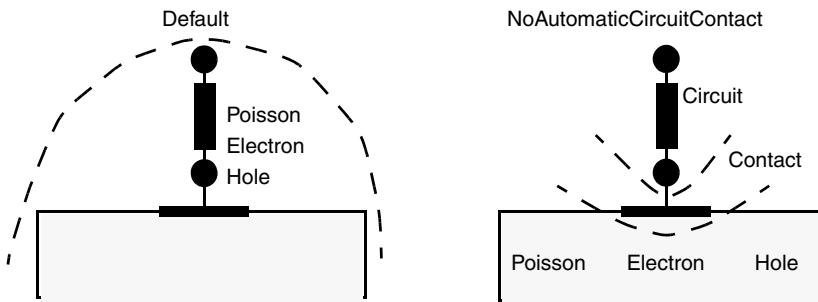


Figure 22 Range of equation-variable keywords Circuit, Contact, Poisson, Electron, and Hole

Selecting Individual Devices

The default usage of an equation-variable keyword such as `Poisson` activates the given equations and variables 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-variable pair to be restricted to a specific device by adding the name of the device instance to the equation-variable keyword separated with a period. The syntax is:

```
<identifier>.<equation-variable>
```

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-variable
    Circuit

    # Solve poisson and full coupled for each device
    Coupled { device1.Poisson device1.Contact }
    Coupled { device1.Poisson device1.Contact }
```

3: Mixed-Mode Sentaurus Device

Solve Section

```
        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 }
}
```

Quasistationary Command

The Quasistationary command is extended in mixed mode to include goals on nodes and circuit model parameters. [Table 134 on page 1101](#) shows the syntax of these goals.

The goal is usually used on a node that has been fixed with the `Set` or `Initialize` command in the `System` section. The keyword `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 keyword `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 1Ω to 0.1Ω :

```
System {
    Resistor_pset r1(a 0){resistance = 1}
    Set(a=1)
}

Solve{
    Circuit
    Quasistationary(Goal{Parameter=r1."resistance" Value=0.1})
        { Circuit }
}
```

NOTE When a node is used in a goal, it is set during the quasistatic 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.

ACCoupled: 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 129 on page 1097](#) describes these parameters. [AC Simulation on page 893](#) provides technical background information for the method.

The options `StartFrequency`, `EndFrequency`, `NumberOfPoints`, `Linear`, and `Decade` are used to select the frequencies at which the analysis is performed and the frequency distribution.

A Node list must be given. With it, for each frequency, the compact equivalent small signal model is generated between the given nodes. This conductive-capacitive matrix is stored in an ACEExtract file specified in the `File` or `ACcoupled` statement (default is `extraction_ac_des.plt`). This ASCII file contains the frequency, voltages at the nodes, and entries of the matrix. If the file name prefix `ACPlot` is specified, for each node in the Node list and for each frequency, Sentaurus Device writes a file with the (complex-valued) derivatives of the solution variables and currents with respect to voltage at the node.

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` command. The parameters in `ACCompute` are identical to the parameters in the `Plot` and `Save` commands (see [Table 136 on page 1103](#)), 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)
```

3: Mixed-Mode Sentaurus Device

Solve Section

```
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]. Additionally, 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` command behaves like an ordinary `Coupled` command inside a `Quasistationary`.

Example: AC Analysis of Simple Device

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 reverse bias of -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 one frequency $f = 10^3$ Hz. The circuit element `drive` and `to_ground` are excluded from the AC calculation.

By including circuits, complete Bode plots can be performed:

```
Device resist {
    Electrode {{Name=anode Voltage=-3 resist=1}
               {Name=cathode Voltage=0 resist=1}}
}

File {Grid = "resist"
      Doping = "resist"}

Physics {Mobility (DopingDependence)
         Recombination (SRH(DopingDependence) Auger)}
}

System {
    resist 1d (anode=left cathode=right)
    Vsource_pset drive(left right){dc = -3}
    Resistor_pset to_ground (right 0){resistance=0}
}

Math {
    Method=Blocked SubMethod=Super      # default solvers for Coupled
    ACMETHOD=Blocked ACSUBMETHOD=Super # default solvers for AC analysis
}
```

```

        NoAutomaticCircuitContact
    }

    Solve {
        Poisson
        Coupled {Poisson Electron Hole}
        Coupled {poisson electron hole contact circuit}
        ACCoupled (StartFrequency=1e3 EndFrequency=1e6
                    NumberOfPoints=4 Decade
                    Iterations=0
                    Node(left right)
                    Exclude(drive to_ground)
                    ACMETHOD=Blocked ACSubMethod("1d")=ParDiSo)
                    {poisson electron hole contact circuit}
        Quasistat (MaxStep=0.025 InitialStep=0.025
                    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)
                    {electron hole poisson Circuit Contact}
        }
    }
}

```

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 }

```

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 79](#)), HB is computationally more efficient for problems with time constants that differ by many orders of magnitude. Compared to AC analysis (see [ACCoupled: Small-Signal AC Analysis on page 161](#)), 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 81](#)).

NOTE Harmonic balance is not supported for traps (see [Chapter 10 on page 349](#)) or ferroelectrics (see [Chapter 22 on page 589](#)).

The time-dependent simulation problems take the form:

$$\frac{d}{dt}q[r, u(t, r)] + f[r, u(t, r), w(t, r)] = 0 \quad (22)$$

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 \hat{f}_m , and $H = (H_1, \dots, H_M)^T$, the vector of nonnegative maximal numbers of harmonics H_m , the solution u is approximated by a truncated Fourier series:

$$u(t) = U_0 + \sum_{-H \leq h \leq H} U_h \exp(i\omega_h t) \quad (23)$$

where $\omega_h = 2\pi h \cdot \hat{f}$.

In Fourier space, [Eq. 22](#) becomes:

$$L(U) := i\Omega Q(U) + F(U) = 0 \quad (24)$$

where Ω 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 [Eq. 24](#) requires a nonlinear (Newton) iteration. For more details on the numerics of harmonic balance, see [Harmonic Balance Analysis on page 895](#).

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, CMI Models and Sentaurus Device Analysis Methods on page 117](#).

Sentaurus Device provides a basic set of compact models that support this functional behavior (refer to [Compact Models User Guide, CMI Models with Frequency-Domain Assembly on page 164](#)). Standard SPICE models are not supported in this mode.

SDFT Mode

The SDFT mode supports only one-tone HB analysis. The mixed-mode circuit may 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 Command on page 65](#)). In addition, `HBCoupled` supports the options summarized in [Table 135 on page 1102](#).

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.

3: Mixed-Mode Sentaurus Device

Solve Section

An example of a two-tone analysis is:

```
Math {
    HB { MDFT }      * enable MDFT mode
    ...
}
System {
    ...
    sd_hb_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 )
    ) { 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 may be interested only in a few spectrum components or you may 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 153 on page 1122](#)), others can be set exclusively in the HBCoupled statement (see [Table 135 on page 1102](#)), and some can be set in both places.

`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.

3: Mixed-Mode Sentaurus Device

Solve Section

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.

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 \tilde{U}_h of:

$$\begin{aligned} u(t) &= U_0 + \sum_{\underline{h} \in K^+} \operatorname{Re}(U_{\underline{h}} \exp(i\omega_{\underline{h}} t)) \\ &= \tilde{U}_0 + \sum_{\underline{h} \in K^+} \{\operatorname{Re}(\tilde{U}_{\underline{h}}) \cos(\omega_{\underline{h}} t) - \operatorname{Im}(\tilde{U}_{\underline{h}}) \sin(\omega_{\underline{h}} t)\} \end{aligned} \quad (25)$$

where the sum is taken over all multi-indices \underline{h} , which results in a positive frequency $\omega_{\underline{h}} = \underline{h} \cdot \underline{\omega}$.

Device Instance Currents, Voltages, Temperatures, and Heat Components

Each converged `HBCoupled` plots the results (contact currents and voltages, temperatures, and heat components) both in the time domain and frequency domain into a separate file. The names of the files for the time domain contain a component `Tdom`; those for the frequency domain contain a component `Hdom`.

Circuit Currents and Voltages

Additionally, 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 [System Plot on page 155](#)). In MDFT mode, each converged `HBCoupled` plots its results in a `Tdom` and an `Hdom` file as for device instances, while in SDFT mode, it will write four files containing all Fourier coefficients (files with the name `component Fdomain`), the time domain data (`Tdomain`), the harmonic magnitude (`Hmag`), and the harmonic phase (`Hphase`).

Solution Variables

Plotting solution variables is only supported for one-tone HB simulations. This is implicitly done if the `HB` section in the `Math` section is present. Sentaurus Device will plot the coefficients \tilde{U}_h of [Eq. 25](#) as a real-valued vector with components $U_0 = \tilde{U}_0$, $U_{2h-1} = \operatorname{Re}(\tilde{U}_h)$, and $U_{2h} = \operatorname{Im}(\tilde{U}_h)$ (for $1 \leq h \leq 3$), with names composed of a prefix `HB`, a suffix `_C<i>` with component `<i>`, and the names of the solution variables. Additionally, the magnitude and phase of \tilde{U}_h (for $0 \leq h \leq 3$) are plotted, with the prefixes `Mag` and `Phase` to the variable names and suffixes `_C<h>`.

Application Notes

- 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.

Set and Unset Section

The keyword `Set` in the `Solve` section is used to set node values during a part of the simulation. The `Set` command in the `Solve` section is position dependent. This allows a node to be set to a previously computed value. The `Set` command 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` command with the specified node.

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

The `Set` command can also be used to affect the boundary condition type of electrodes in a single device mode (see [Set Command on page 91](#)).

Accessing SPICE Vector Parameters

The SPICE voltage and current sources use vector parameters to define the properties of various source types. For example:

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

defines a voltage source with an offset $v_0 = \text{sine}[0] = 0.5$ V, an amplitude $v_a = \text{sine}[1] = 1$ V, a frequency $\text{freq} = \text{sine}[2] = 10$ Hz, a delay $\text{td} = \text{sine}[3] = 0$ s, and a damping factor $\text{theta} = \text{sine}[4] = 0\text{s}^{-1}$. Components of such vectors can be set or ramped in the `Solve` section. As an example, consider:

```
Solve {
    Set (v0."sine[0]" = 0.75)
    Quasistationary (Goal {Parameter = v0."sine[0]" Value = 1.5}) { Circuit }
}
```

3: Mixed-Mode Sentaurus Device

Math Section

First, the offset of the sine voltage source v0 is set to 0.75 V. Afterwards, the offset is ramped to 1.5 V. In a similar way, the components of vector parameters can be plotted in a system plot:

```
System {
    Plot (p(v0 "sine[0]"))
}
```

Math Section

The keyword `AutomaticCircuitContact` (active by default) controls the automatic inclusion of the circuit and contact equations in a mixed-mode simulation. If only the Poisson equation is solved, no additional equations are added. However, if additional equations to Poisson appear in a `Coupled` statement, the circuit and contact equations are also added.

Therefore:

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

is equivalent to:

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

NOTE `AutomaticCircuitContact` does not add the 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 [Circuit and Contact Equation–Variable Keywords on page 158](#)). The following SPICE circuit parameters can be specified in the global `Math` section:

```
Spice_Temperature = ...
Spice_gmin = ...
```

The value of `Spice_Temperature` denotes the global SPICE circuit temperature. Its default value is 300.15 K. The parameter `Spice_gmin` refers to the minimum conductance in SPICE. The default value is $10^{-12} \Omega^{-1}$.

Using Mixed-Mode Simulation

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

From Single-Device File to Multidevice File

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 input 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 23](#)). The three levels are linked in that the global level is the default for the device level and instance level.

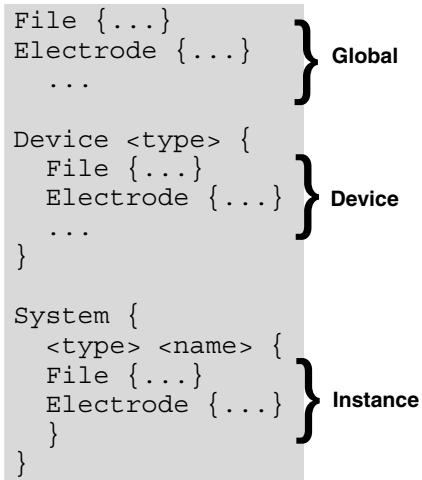


Figure 23 Three levels of device definition

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).

3: Mixed-Mode Sentaurus Device

Using Mixed-Mode Simulation

This translation process can be performed manually by creating a **Device** and **System** section with a single entry (see [Figure 24](#)).

```
Electrode {  
    {name="anode" Voltage=0}  
    {name="cathode" Voltage=1}  
}  
  
File {  
    Grid = "diode.tdr"  
    Doping = "diode.tdr"  
    Output = "diode"  
}  
  
File {  
    Output = "diode"  
}  
Device diode {  
    Electrode {  
        {name="anode" Voltage=0}  
        {name="cathode" Voltage=1}  
    }  
    File {  
        Grid = "diode.tdr"  
        Doping = "diode.tdr"  
    }  
}  
System {  
    diode diode1 ...  
}
```

Figure 24 Translating a single device syntax to mixed-mode form

The **Solve** section can also be converted if the flag **NoAutomaticCircuitContact** is used (see [Figure 25](#)).

```
Math {  
    NoAutomaticCircuitContact  
}  
Solve {  
    Coupled {Poisson Contact Circuit}  
    Coupled {Poisson Contact Circuit Electron Hole}  
}  
  
Solve {  
    Poisson  
    Coupled {Poisson Electron Hole}  
}
```

Figure 25 Translating a default solve syntax to a **NoAutomaticCircuitContact** form

In this case, all occurrences of the keyword **Poisson** must be expanded to the three keywords **Poisson Contact Circuit**.

File-naming Convention: Mixed-Mode Extension

A `File` section can be defined at all levels of an input command file. Therefore, a default file name can be potentially included by more than one device, for example:

```
Device res {  
    File { Save="res" ... }  
    ...  
}  
System {  
    res r1  
    res r2  
}
```

Both `r1` and `r2` use the save device parameters set up in the definition of `res`. Therefore, they have in principle the same default for `Save` (that is, `res`). Since it is impractical to save both devices under the same name, the names of the 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 is performed for the file parameters `Save`, `Current`, `Path`, and `Plot`, and is also performed when the file name is copied from the global default to a device type declaration.

In practice, three 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 type level, in which case, the instance name is concatenated to the original file name.
- The file name is defined at the global input file level, in which case, the device name and instance name are concatenated to the original file name.

[Table 20](#) summarizes these possibilities.

Table 20 File modification for Save, Current, Path, and Plot commands

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

3: Mixed-Mode Sentaurus Device

Using Mixed-Mode Simulation

Part II Physics in Sentaurus Device

This part of the Sentaurus Device manual contains the following chapters:

- [Chapter 4 Introduction to Physics in Sentaurus Device on page 177](#)
- [Chapter 5 Semiconductor Band Structure on page 225](#)
- [Chapter 6 Incomplete Ionization on page 245](#)
- [Chapter 7 Quantization Models on page 251](#)
- [Chapter 8 Mobility Models on page 267](#)
- [Chapter 9 Generation–Recombination on page 309](#)
- [Chapter 10 Traps and Fixed Charges on page 349](#)
- [Chapter 11 Phase and State Transitions on page 365](#)
- [Chapter 12 Degradation Model on page 373](#)
- [Chapter 13 Organic Devices on page 389](#)
- [Chapter 14 Optical Generation on page 393](#)
- [Chapter 15 Radiation Models on page 489](#)
- [Chapter 16 Noise and Fluctuation Analysis on page 499](#)
- [Chapter 17 Tunneling on page 511](#)
- [Chapter 18 Hot-Carrier Injection Models on page 531](#)
- [Chapter 19 Heterostructure Device Simulation on page 547](#)
- [Chapter 20 Energy-dependent Parameters on page 565](#)
- [Chapter 21 Anisotropic Properties on page 575](#)
- [Chapter 22 Ferroelectric Materials on page 589](#)
- [Chapter 23 Modeling Mechanical Stress Effect on page 595](#)
- [Chapter 24 Galvanic Transport Model on page 643](#)
- [Chapter 25 Thermal Properties on page 647](#)

Introduction to Physics in Sentaurs Device

This chapter presents an overview of physics as it applies to the functionality of Sentaurs Device.

Physical phenomena in semiconductor devices are very complicated and, depending on applications, are described by partial differential equations of different level of complexity. Coefficients and boundary conditions of equations (such as mobility, generation–recombination rate, material-dependent parameters, interface and contact boundary conditions) can be very complicated and can depend on microscopic physics, the structure of the device, and the applied bias.

Sentaurs Device allows for arbitrary combinations of transport equations and physical models, which allows for the possibility to simulate all spectrums of semiconductor devices, from power devices to deep submicron devices and sophisticated heterostructures. This chapter describes the formulation of physical models and equations.

Transport Equations

Depending on the device under investigation and the level of modeling accuracy required, you can select four different simulation modes:

- Drift-diffusion
 - Isothermal simulation, described by basic semiconductor equations. Suitable for low-power density devices with long active regions.
- Thermodynamic
 - Accounts for self-heating. Suitable for devices with low thermal exchange, particularly, high-power density devices with long active regions.
- Hydrodynamic
 - Accounts for energy transport of the carriers. Suitable for devices with small active regions.
- Monte Carlo
 - Allows for full band Monte Carlo device simulation in the selected window of the device.

4: Introduction to Physics in Sentaurus Device

Transport Equations

The equations for the drift-diffusion, thermodynamic, and hydrodynamic modes are presented in this section.

The Monte Carlo mode allows the Boltzmann transport equation to be solved in a selected window of the device. Monte Carlo simulation is provided by the device simulator Sentaurus SPARTA (refer to the *Sentaurus Device Monte Carlo User Guide* for more information).

The equations are formulated initially under the assumption of Boltzmann statistics for electrons and holes. See [Fermi Statistics on page 202](#) for a description of the corresponding equations for Fermi–Dirac statistics.

The transport model can be selected independently for either carrier in Sentaurus Device, or transport can be neglected by assuming a constant quasi-Fermi level for a nonselected carrier. The same applies for the energy balance equation. If it is solved for the temperature of one carrier only, the temperature of the other carrier is assumed to be equal to the lattice temperature. The **Solve** section specifies the set of equations to be solved (see [Solve Section on page 64](#)).

Poisson Equation and Continuity Equations

The three governing equations for charge transport in semiconductor devices are the Poisson equation and the electron and hole continuity equations. The Poisson equation is:

$$\nabla \cdot \epsilon \nabla \phi = -q(p - n + N_D - N_A) - \rho_{\text{trap}} \quad (26)$$

where ϵ is the electrical permittivity, q is the elementary electronic charge, n and p are the electron and hole densities, N_D is the concentration of ionized donors, N_A is the concentration of ionized acceptors, and ρ_{trap} is the charge density contributed by traps and fixed charges (see [Chapter 10 on page 349](#)).

The keyword for the Poisson equation is `Poisson`. The keywords for the electron and hole continuity equations are `electron` and `hole`, respectively. They are written as:

$$\nabla \cdot \vec{J}_n = qR_{\text{net}} + q\frac{\partial n}{\partial t} \quad -\nabla \cdot \vec{J}_p = qR_{\text{net}} + q\frac{\partial p}{\partial t} \quad (27)$$

where R_{net} is the net electron–hole recombination rate (see [Chapter 9 on page 309](#)), \vec{J}_n is the electron current density, and \vec{J}_p is the hole current density.

Drift-Diffusion Model

The drift-diffusion model is widely used for the simulation of carrier transport in semiconductors and is defined by the basic semiconductor equations (see [Poisson Equation and Continuity Equations on page 178](#)), where current densities for electrons and holes are given by:

$$\vec{J}_n = -nq\mu_n \nabla \Phi_n \quad (28)$$

$$\vec{J}_p = -pq\mu_p \nabla \Phi_p \quad (29)$$

where μ_n and μ_p are the electron and hole mobilities (see [Chapter 8 on page 267](#)), and Φ_n and Φ_p are the electron and hole quasi-Fermi potentials, respectively (see [Quasi-Fermi Potential on page 201](#)).

Thermodynamic Transport Model

The thermodynamic (or nonisothermal) model [1] extends the drift-diffusion approach to account for electrothermal effects, under the assumption that the charge carriers are in thermal equilibrium with the lattice. Therefore, the carrier temperatures and the lattice temperature are described by a single temperature T .

Thermodynamic Model

The thermodynamic model is defined by the basic set of partial differential equations [Eq. 26](#) and [Eq. 27](#), and the lattice heat flow equation [Eq. 32](#). The relations [Eq. 28](#) and [Eq. 29](#) are generalized to include the temperature gradient as a driving term:

$$\vec{J}_n = -nq\mu_n(\nabla \Phi_n + P_n \nabla T) \quad (30)$$

$$\vec{J}_p = -pq\mu_p(\nabla \Phi_p + P_p \nabla T) \quad (31)$$

where P_n and P_p are the absolute thermoelectric powers [2] (see [Thermoelectric Power \(TEP\) on page 649](#)). For accurate simulation of self-heating effects, this extra driving force for the current can be included by specifying the keyword `Thermodynamic` in the `Physics` section of the input file.

4: Introduction to Physics in Sentaurus Device

Transport Equations

To calculate the temperature distribution in the device due to self-heating, the following equation is solved:

$$\frac{\partial}{\partial t} c_L T - \nabla \cdot \kappa \nabla T = -\nabla \cdot [(P_n T + \Phi_n) \vec{J}_n + (P_p T + \Phi_p) \vec{J}_p] - \left(E_C + \frac{3}{2} kT \right) \nabla \cdot \vec{J}_n - \left(E_V - \frac{3}{2} kT \right) \nabla \cdot \vec{J}_p + q R_{\text{net}} (E_C - E_V + 3kT) \quad (32)$$

where κ is the thermal conductivity (see [Thermal Conductivity on page 648](#)) and c_L is the lattice heat capacity (see [Heat Capacity on page 647](#)). E_C and E_V are the conduction and valence band energies, respectively, and R_{net} is the recombination rate.

The Peltier effect at metal–semiconductor interfaces can also be accounted for by adding to the right-hand side of [Eq. 32](#) the corresponding electron and hole Peltier heat on the interface. Peltier heat at a metal–semiconductor interface is modeled as:

$$Q = J_n (\alpha_n \Delta E_n + (1 - \alpha_n) \Delta \varepsilon_n) \quad (33)$$

$$Q = J_p (\alpha_p \Delta E_p + (1 - \alpha_p) \Delta \varepsilon_p) \quad (34)$$

where Q is the heat density at the interface (when $Q > 0$, there is heating; when $Q < 0$, cooling), J_n and J_p are the electron and hole current densities normal to the interface, ΔE_n and ΔE_p are the energy differences for electrons and holes across the interface, and α_n , α_p , $\Delta \varepsilon_n$, and $\Delta \varepsilon_p$ are fitting parameters with $0 \leq \alpha_n, \alpha_p \leq 1$.

Sentaurus Device computes the energy differences for electrons and holes as $\Delta E_n = \Phi_M - \beta_n \Phi_n + (1 - \beta_n) E_C / q$ and $\Delta E_p = \Phi_M - \beta_p \Phi_p + (1 - \beta_p) E_V / q$, respectively, where β_n and β_p are fitting parameters.

Using the Thermodynamic Model

To activate a nonisothermal simulation, the keyword `Temperature` must be specified inside the `Coupled` command of the `Solve` section (see [Solve Section on page 64](#)). To activate extra terms in the current density equations (due to the gradient of the temperature), the keyword `Thermodynamic` must be specified in the `Physics` section. Without `Thermodynamic` (and `Temperature` still included in the `Solve` section), Sentaurus Device uses [Eq. 36](#) and [Eq. 37](#) instead of [Eq. 30](#) and [Eq. 31](#), and uses [Eq. 40](#) instead of [Eq. 32](#) (see [Hydrodynamic Transport Model on page 184](#)).

To activate Peltier heat at a metal–semiconductor interface, the keyword `MSPeltierHeat` must be specified inside the corresponding interface `Physics` section:

```
Physics(MaterialInterface="Silicon/Metal") {
    MSPeltierHeat
    ...
}
```

The two fitting parameters α and $\Delta\epsilon$ in the parameter file can be specified in the `MSPeltierHeat` section of the region interface for which the Peltier heat is computed:

```
MaterialInterface = "Silicon/Metal" {
    MSPeltierHeat
    {
        alpha  = 1.0 , 1.0 # [1]
        beta   = 1.0 , 1.0
        deltaE = 0.0 , 0.0 # [eV]
    }
}
```

Their default values are $\alpha_n = \alpha_p = \beta_n = \beta_p = 1$ and $\Delta\epsilon_n = \Delta\epsilon_p = 0$ eV.

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 [Eq. 32](#). It is plotted using the `TotalHeat` keyword in the `Plot` section. The total heat is calculated only when Sentaurus Device solves the temperature equation.

The formulas to estimate individual heating mechanisms and the appropriate keywords for use in the `Plot` section of the command file (see [Plot Section on page 58](#)) are shown in [Table 21](#). 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.

Table 21 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	$\frac{ \vec{J}_p ^2}{qp\mu_p}$
Recombination heat	RecombinationHeat	$qR_{\text{net}}(\Phi_p - \Phi_n + T(P_p - P_n))$
Thomson plus Peltier heat [3]	ThomsonHeat	$-\vec{J}_n \cdot T\nabla P_n - \vec{J}_p \cdot T\nabla P_p$

4: Introduction to Physics in Sentaurus Device

Transport Equations

Table 21 Terms and keywords used in Plot section of command file

Heat name	Keyword	Formula
Peltier heat	PeltierHeat	$-T \left(\frac{\partial P_n}{\partial n} \vec{J}_n \cdot \nabla n + \frac{\partial P_p}{\partial p} \vec{J}_p \cdot \nabla p \right)$

To plot the lattice heat flux vector $\kappa \nabla T$, specify the keyword `1HeatFlux` in the `Plot` section (see [Plot Section on page 58](#)).

Uniform Self-Heating

A simplified self-heating model is also 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 or current dependent. The global temperature is computed from a global heat balance equation where the dissipated power P_{diss} is equal to the sum of the boundary heat fluxes (on thermodes with finite thermal resistance):

$$P_{\text{diss}} = \sum_i \frac{T - T_{\text{thermode}}^{(i)}}{R_{\text{th}}^{(i)}} \quad (35)$$

where T is the device global temperature, and $T_{\text{thermode}}^{(i)}$ and $R_{\text{th}}^{(i)}$ are the temperature and thermal resistivity of thermode i , respectively. If a thermode does not specify a thermal resistance or a thermal conductance, then by default, a zero surface conductance is assumed. In this case, the thermode is not accounted for in the sum on the right-hand side of [Eq. 35](#) because the respective heat flux is zero. In the extreme case, when none of the thermodes specifies any thermal resistance or conductance, [Eq. 35](#) cannot be solved for temperature.

The dissipated power P_{diss} can be calculated as either the sum of all terminal currents multiplied by their respective terminal voltages ($\sum IV$), the sum of IV s from the user-specified node, or the total Joule heat $\int_V (\vec{J} \cdot \vec{F}) dv$.

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 This uniform self-heating feature is a postprocessing one where the temperature is not computed self-consistently with all other solution variables. Therefore, the simulation results may depend on the simulation time step used in quasistationary or transient simulations. In addition, this feature must not be used for AC, noise, or harmonic balance (HB) simulations.

Using Uniform Self-Heating

The simplified 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_{diss} is computed can be chosen by the syntax of PostTemperature in the command file:

- a) As $\sum IV$ over all electrodes:

```
Physics {
    PostTemperature(IV_diss)
    ...
}
```

- b) As $\sum IV$ at user-selected electrodes:

```
Physics {
    PostTemperature(IV_diss("contact1" "contact2"))
    ...
}
```

- c) As the integral of Joule heat over the device volume:

```
Physics {
    PostTemperature
    ...
}
```

This feature can be used during a quasistationary or transient simulation. In addition, because the feature replaces the lattice heat flow equation ([Eq. 32, p. 180](#)), it cannot be used with the lattice temperature equation activated in the Coupled section.

Hydrodynamic Transport Model

With continued scaling into the deep submicron regime, neither internal nor external characteristics of state-of-the-art semiconductor devices can 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. The Monte Carlo method for the solution of the Boltzmann kinetic equation is the most general approach, but because of its high computational requirements, it cannot be used for the routine simulation of devices in an industrial setting.

In this case, the hydrodynamic (or energy balance) model provides a very good compromise. Since the work of Stratton [4] and Bløtekjær [5], there have been many variations of this model [6]–[20]. The full formulation, including the so-called convective terms [7], consists of eight partial differential equations (PDEs) [8][21][22], while the simpler form (no convective terms) includes six PDEs [9]–[11][19].

Using the Hydrodynamic Model

To perform a hydrodynamic simulation, the keyword `eTemperature` (synonym `ElectronTemperature`) or `hTemperature` (synonym `HoleTemperature`) must be specified inside the `Coupled` command of the `Solve` section (see [Solve Section on page 64](#)).

In addition, to activate the hydrodynamic model, the keyword `Hydrodynamic` (or `Hydro`) must be specified in the `Physics` section. If only one carrier temperature equation is to be solved, `Hydro` must be specified with the appropriate parameter, either `Hydro(eTemp)` or `Hydro(hTemp)`. 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 [Eq. 38](#), [Eq. 39](#), and [Eq. 40](#)) are added.

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 [Eq. 41](#), [Eq. 42](#), and [Eq. 43](#)), specify the corresponding keywords `eHeatFlux`, `hHeatFlux`, and `lHeatFlux` in the `Plot` section (see [Plot Section on page 58](#)).

Hydrodynamic Model

The transport model implemented in Sentaurus Device is based on the approach involving the solution of six PDEs. In the hydrodynamic model, the carrier temperatures T_n and T_p are

assumed to not equal the lattice temperature T . Together with basic semiconductor equations, up to three additional equations can be solved to find the temperatures. In general, the model consists of the basic set of PDEs (the Poisson equation and continuity equations, see [Poisson Equation and Continuity Equations on page 178](#)) and the energy conservation equations for electrons, holes, and the lattice.

In the hydrodynamic case, current densities are defined as:

$$\vec{J}_n = q\mu_n \left(n\nabla E_C + kT_n \nabla n + f_n^{\text{td}} kn \nabla T_n - 1.5nkT_n \nabla \ln m_n \right) \quad (36)$$

$$\vec{J}_p = q\mu_p \left(p\nabla E_V - kT_p \nabla p - f_p^{\text{td}} kp \nabla T_p - 1.5pkT_p \nabla \ln m_p \right) \quad (37)$$

The first term takes into account the contribution due to the spatial variations of electrostatic potential, electron affinity, and the band gap. The three remaining terms in [Eq. 36](#) and [Eq. 37](#) 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 .

The form of the abovementioned current density equations appears because the well-known Einstein relation is applied. The relation defines the carrier diffusion coefficient as $D = \mu kT$, but it is known that this is valid only near the equilibrium. Therefore, Sentaurus Device provides an option for you to modify this relation and, thereby, to modify the carrier diffusion. This option allows you to represent the carrier temperature used in [Eq. 36](#) and [Eq. 37](#) as a superposition of a solution for the carrier temperature T_c and the lattice temperature T . This can be useful in devices where the carrier diffusion is important. The new temperature expressed as $gT_c + (1-g)T$ replaces all occurrences of T_n and T_p in [Eq. 36](#) and [Eq. 37](#), respectively. The coefficient g for electrons and holes can be specified in the ThermalDiffusion parameter set.

The energy balance equations read:

$$\frac{\partial W_n}{\partial t} + \nabla \cdot \vec{S}_n = \vec{J}_n \cdot \nabla E_C + \frac{dW_n}{dt} \Big|_{\text{coll}} \quad (38)$$

$$\frac{\partial W_p}{\partial t} + \nabla \cdot \vec{S}_p = \vec{J}_p \cdot \nabla E_V + \frac{dW_p}{dt} \Big|_{\text{coll}} \quad (39)$$

$$\frac{\partial W_L}{\partial t} + \nabla \cdot \vec{S}_L = \frac{dW_L}{dt} \Big|_{\text{coll}} \quad (40)$$

4: Introduction to Physics in Sentaurus Device

Transport Equations

where the energy fluxes are:

$$\vec{S}_n = -\frac{5r_n}{2} \left(\frac{kT_n}{q} \vec{J}_n + f_n^{\text{hf}} \hat{\kappa}_n \nabla T_n \right) \quad (41)$$

$$\vec{S}_p = -\frac{5r_p}{2} \left(\frac{-kT_p}{q} \vec{J}_p + f_p^{\text{hf}} \hat{\kappa}_p \nabla T_p \right) \quad (42)$$

$$\vec{S}_{\text{L}} = -\kappa_{\text{L}} \nabla T_{\text{L}} \quad (43)$$

$$\hat{\kappa}_n = \frac{k^2}{q} n \mu_n T_n \quad (44)$$

$$\hat{\kappa}_p = \frac{k^2}{q} p \mu_p T_p \quad (45)$$

The parameters r_n , r_p , f_n^{d} , f_p^{d} , f_n^{hf} , and f_p^{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/energy balance models as described in the literature, from Stratton to Bløtekjær [4][5][15][19][20]. The default parameter values of Sentaurus Device are:

$$r_n = r_p = 0.6 \quad (46)$$

$$f_n^{\text{d}} = f_p^{\text{d}} = 0 \quad (47)$$

$$f_n^{\text{hf}} = f_p^{\text{hf}} = 1 \quad (48)$$

By changing the constants f_n^{hf} and r_n , the convective contribution and the diffusive contributions can be changed independently. With the default set of transport parameters of Sentaurus Device, the prefactor of the diffusive term has the form:

$$\kappa_n = \frac{3}{2} \cdot \frac{k^2}{q} n \mu_n T_n \quad (49)$$

If the hydrodynamic model is based on the Bløtekjær approach, the parameters should be:

$$f_n^d = f_p^d = f_n^{hf} = f_p^{hf} = 1 \quad , \quad r_n = r_p = 1 \quad (50)$$

The collision terms are expressed by this set of equations:

$$\frac{\partial W_n}{\partial t} \Big|_{\text{coll}} = -H_n - \xi_n \frac{W_n - W_{n0}}{\tau_{en}} \quad (51)$$

$$\frac{\partial W_p}{\partial t} \Big|_{\text{coll}} = -H_p - \xi_p \frac{W_p - W_{p0}}{\tau_{ep}} \quad (52)$$

$$\frac{\partial W_L}{\partial t} \Big|_{\text{coll}} = H_L + \xi_n \frac{W_n - W_{n0}}{\tau_{en}} + \xi_p \frac{W_p - W_{p0}}{\tau_{ep}} \quad (53)$$

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 [11] and have the following form for the major generation–recombination phenomena:

$$H_n = 1.5kT_n(R_{\text{net}}^{\text{SRH}} + R_{\text{net}}^{\text{rad}} + R_{n,\text{net}}^{\text{trap}}) - E_{g,\text{eff}}(R_n^A - G_n^{\text{ii}}) - \alpha(\hbar\omega - E_{g,\text{eff}})G^{\text{opt}} \quad (54)$$

$$H_p = 1.5kT_p(R_{\text{net}}^{\text{SRH}} + R_{\text{net}}^{\text{rad}} + R_{p,\text{net}}^{\text{trap}}) - E_{g,\text{eff}}(R_p^A - G_p^{\text{ii}}) - (1 - \alpha)(\hbar\omega - E_{g,\text{eff}})G^{\text{opt}} \quad (55)$$

$$H_L = [R_{\text{net}}^{\text{SRH}} + 0.5(R_{n,\text{net}}^{\text{trap}} + R_{p,\text{net}}^{\text{trap}})](E_{g,\text{eff}} + 1.5kT_n + 1.5kT_p) \quad (56)$$

where:

- $R_{\text{net}}^{\text{SRH}}$ is the Shockley–Read–Hall (SRH) recombination rate (see [Shockley–Read–Hall Recombination on page 309](#)).
- $R_{\text{net}}^{\text{rad}}$ is the radiative recombination rate (see [Radiative Recombination on page 321](#)).
- R_n^A and R_p^A are Auger recombination rates (see [Auger Recombination on page 322](#)) related to electrons and holes.
- G_n^{ii} and G_p^{ii} are impact ionization rates (see [Avalanche Generation on page 324](#)).
- $R_{n,\text{net}}^{\text{trap}}$ and $R_{p,\text{net}}^{\text{trap}}$ are the recombination rates through trap levels (see [Chapter 10 on page 349](#)).
- G^{opt} is the optical generation rate (see [Chapter 14 on page 393](#)).

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.

4: Introduction to Physics in Sentaurus Device

Transport Equations

The energy densities W_n , W_p , and W_L are given by:

$$W_n = nw_n = n\left(\frac{3kT_n}{2}\right) \quad (57)$$

$$W_p = pw_p = p\left(\frac{3kT_p}{2}\right) \quad (58)$$

$$W_L = c_L T \quad (59)$$

and the corresponding equilibrium energy densities are:

$$W_{n0} = nw_0 = n\frac{3kT}{2} \quad (60)$$

$$W_{p0} = pw_0 = p\frac{3kT}{2} \quad (61)$$

The parameters ξ_n and ξ_p in [Eq. 51](#) to [Eq. 53](#) improve numeric stability. They speed up relaxation for small densities and they approach 1 for large densities:

$$\xi_n = 1 + \frac{n_{\min}}{n} \left(\frac{n_0}{n_{\min}} \right)^{\max[0, (T - T_n)/100 \text{ K}]} \quad (62)$$

and likewise for ξ_p . Here, n_{\min} and n_0 are adjustable small density parameters.

If the hydrodynamic model is only used for one carrier, the temperature of the other carrier is set to the lattice temperature. In this case, the hydrodynamic equations are reformulated to reflect this. For example, if the hydrodynamic option is only applied to the electrons, the Joule heat generated by the hole current is directly captured in the lattice temperature equation.

Hydrodynamic Model Parameters

The default set of transport coefficients ([Eq. 46](#), p. 186 to [Eq. 48](#), p. 186) can be changed in the parameter file. The coefficients r , f^{ad} , and f^{hf} are specified in the EnergyFlux, ThermalDiffusion, and HeatFlux parameter sets, respectively. Energy relaxation times τ_{en} and τ_{ep} can be modified in the EnergyRelaxationTime parameter set.

α in [Eq. 54](#) and [Eq. 55](#) divides the contribution of the optical generation rate into the energy gain or loss terms H_n and H_p . OptGenOffset specifies α , which can take values between 0 and 1 (default is 0.5). The angular frequency ω in [Eq. 54](#) and [Eq. 55](#) 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 [Loading Optical](#)

[Generation from File on page 483](#)). OptGenOffset and OptGenWavelength are both options to RecGenHeat (see [Table 159 on page 1125](#)).

n_{\min} and n_0 in [Eq. 62](#) are set with the parameters RelTermMinDensity and RelTermMinDensityZero, respectively, in the global Math section. The default values for n_{\min} and n_0 are 10^3 cm^{-3} and $2 \times 10^8 \text{ cm}^{-3}$, respectively.

Singlet Exciton Equation

In organic semiconductor devices, besides the simulation of the electrical part consisting of solving the Poisson and continuity equations, a new equation – the singlet exciton equation – is introduced to account for optical properties of these materials using Frenkel excitons.

This equation takes into account only singlet excitons as 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:

$$\frac{\partial n_{\text{se}}}{\partial t} = R_{\text{bimolec}} + D_{\text{se}} \nabla \cdot \nabla n_{\text{se}} - \frac{n_{\text{se}} - n_{\text{se}}^{\text{eq}}}{\tau} - \frac{n_{\text{se}} - n_{\text{se}}^{\text{eq}}}{\tau_{\text{trap}}} - R_{\text{se}} \quad (63)$$

where:

- n_{se} is the singlet exciton density.
- R_{bimolec} is the carrier bimolecular recombination rate acting as a singlet exciton generation term.
- D_{se} is the singlet exciton diffusion constant.
- τ, τ_{trap} are the singlet exciton lifetimes.
- R_{se} is the net singlet exciton recombination rate.

The first term in [Eq. 63](#) accounts for the creation of excitons, and this is a direct consequence of carrier bimolecular recombination (see [Bimolecular Recombination on page 344](#)). Exciton transport occurs using the second term (diffusion term) in [Eq. 63](#), while radiative decay associated with light emission is accounted for by the third and fourth terms.

4: Introduction to Physics in Sentaurus Device

Transport Equations

Nonradiative exciton destruction and conversion of excitons to free electron and free hole populations are described by the net exciton recombination rate (fifth term):

$$\begin{aligned} R_{\text{se}} &= R_{\text{se}-nf} + R_{\text{se}-pf} \\ R_{\text{se}-nf} &= (v_{\text{se}} + v_n) \sigma_{\text{se}-n} n_{\text{se}} n \\ R_{\text{se}-pf} &= (v_{\text{se}} + v_p) \sigma_{\text{se}-p} n_{\text{se}} p \end{aligned} \quad (64)$$

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 $v_{p,0}$ can be 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.

Boundary Conditions for Singlet Exciton Equation

At electrodes and thermodes, equilibrium is assumed for a singlet exciton population:

$$n_{\text{se}}(T) = n_{\text{se}}^{\text{eq}}(T) = \gamma g_{\text{ex}}(N_C(T) + N_V(T)) \exp\left(-\frac{E_{\text{g,eff}} - E_{\text{ex}}}{kT}\right) \quad (65)$$

where $\gamma = 1/4$, and g_{ex} and E_{ex} are the singlet exciton degeneracy factor and binding energy, respectively.

For interfaces, boundary conditions for the singlet exciton equation become more elaborate depending on the interface type. The interface type and the corresponding boundary conditions are 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 like the ones in [Eq. 65](#) (default) or $\nabla n_{\text{se}} \cdot \hat{n} = 0$ (zero flux), depending on the user selection.

For a heterointerface where the singlet exciton equation is solved in both regions, boundary conditions imposed are thermionic-like. In this case, you can select the barrier ΔE to be the difference between the band gaps, conduction bands, or valence bands of the two regions.

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:

$$\begin{aligned} j_{se,1} &= j_{se,2} \\ j_{se,2} &= v_{si} \left(n_{se,2} - n_{se,1} \exp\left(-\frac{\Delta E}{kT}\right) \right) \end{aligned} \quad (66)$$

where $n_{se,2}$ and $n_{se,1}$ are the singlet exciton densities of materials 2 and 1 at the heterointerface. 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 [Solve Section on page 64](#)). 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 [Eq. 63, p. 189](#) 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 177 on page 1146](#)).

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`. An example is:

```
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 [Eq. 65, p. 190](#). 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)
}
```

4: Introduction to Physics in Sentaurus Device

Transport Equations

For heterointerfaces, the default barrier type for the singlet exciton flux injection across the interface (defined in [Eq. 66](#)) is the bandgap energy difference.

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 22](#) lists these parameters with their default values.

Table 22 Singlet exciton equation parameters

Symbol	Parameter name	Default value	Location	Unit
γ	gamma	0.25	region	1
τ	tau	1×10^{-7}	region	s
τ_{trap}	tau_trap	1×10^{-8}	region	s
L_{diff}	l_diff	1×10^{-3}	region	cm
$D_{\text{se}} = L_{\text{diff}}^2 / \tau$			region	cm ² /s
$\sigma_{\text{se}-n}$	ex_cXsection	1×10^{-8}	region	cm ²
$\sigma_{\text{se}-p}$	ex_cXsection	1×10^{-8}	region	cm ²
E_{ex}	Eex	0.015	region	eV
g_{ex}	gex	4	region	1
$v_{\text{se},0}$	vth	1×10^7	region	cm/s
$v_{n,0}$	vth_car	1×10^3	region	cm/s
$v_{p,0}$	vth_car	1×10^3	region	cm/s
v_{si}	vel	1×10^8	interface	cm/s

Conductivity of Metals

The simulation of conductivity in metals or semi-metals is important for interconnection problems in ICs. The current density in metals is given by:

$$\vec{J}_M = -q\sigma\nabla\Phi_M \quad (67)$$

where σ is the metal conductivity and Φ_M is the Fermi potential in the metal. For the steady state case, $\nabla \cdot \vec{J}_M = 0$. Therefore, the equation for the Fermi potential inside of metals is:

$$\nabla \cdot \sigma\nabla\Phi_M = 0 \quad (68)$$

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 in insulators and semiconductors to ensure current conservation.
- For interfaces between metal and insulator, the equations are:

$$\begin{aligned} \vec{J}_M \cdot \hat{n} &= \vec{J}_D \cdot \hat{n} \\ \phi &= \Phi_M - \Phi_{MS} \end{aligned} \quad (69)$$

where Φ_{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. It is clear that the electrostatic potential inside metals is computed as $\phi = \Phi_M - \Phi_{MS}$. This is important if, for example, there is a MOSFET with a metal gate.

- For metal–semiconductor interfaces, by default, there is an Ohmic boundary condition that can be written as:

$$\begin{aligned} \vec{J}_M \cdot \hat{n} &= (\vec{J}_n + \vec{J}_p + \vec{J}_D) \cdot \hat{n} \\ \phi &= \Phi_M + \phi_0 \\ n &= n_0 \\ p &= p_0 \end{aligned} \quad (70)$$

ϕ_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 206](#)).

There are several options for the Schottky interface (refer to the equations in [Schottky Contacts on page 208](#)) where the potential barrier between metal and semiconductor will be computed automatically and the barrier tunneling (see [Nonlocal Tunneling at Interfaces, Contacts, and](#)

4: Introduction to Physics in Sentaurus Device

Transport Equations

Junctions on page 518) and barrier lowering (see [Barrier Lowering at Schottky Contacts on page 209](#)) models can be applied.

To select these models, use the following syntax in the command file:

```
Physics(MaterialInterface = "Metal/Silicon") { Schottky eRecVelocity=1e6  
hRecVelocity=1e6 }  
Physics(MaterialInterface = "Metal/Silicon") { Schottky BarrierLowering }  
Physics(MaterialInterface = "Metal/Silicon") { Schottky  
Recombination(eBarrierTunneling) }
```

The default values of the 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, 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.

Both Ohmic and Schottky interfaces support a distributed resistance model similar to the one described in [Resistive Contacts on page 210](#). If a distributed resistance is specified at the interface, a distributed voltage drop $\Delta\phi(R_d) = R_d(\vec{J}_n + \vec{J}_p + \vec{J}_D) \cdot \hat{n}$ 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") {DistResistance=1e-6}  
# Distributed resistance at Schottky interface  
Physics(MaterialInterface = "Metal/Silicon") {Schottky DistResistance=1e-6}  
# Distributed resistance at Ohmic interface emulating a Schottky interface  
Physics(MaterialInterface = "Metal/Silicon")  
{DistResistance = SchottkyResistance}
```

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 using the keyword `DirectCurrent` in the `Math` section. This option may produce a wrong current at the metal contact, especially when the conductivity of the connected metal region is lower (metal silicides).

The following temperature dependence is applied for $\rho = 1/\sigma$:

$$\rho = \rho_0(1 + \alpha_T(T - 273 \text{ K})) \quad (71)$$

All these resistivity parameters can be specified in the parameter file as:

```
Resistivity {
    * Resist(T) = Resist0 * ( 1 + TempCoef * ( T - 273 ) )
    Resist0 = <value> # [Ohm*cm]
    TempCoef = <value> # [1/K]
}
```

To specify the metal workfunction, use the Bandgap parameter set, for example:

```
Material = "Gold" {
    Bandgap{ WorkFunction = 5 # [eV]
}
```

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 [Eq. 68, p. 193](#) is a part of the Contact equation, which is added automatically by default. If the `NoAutomaticCircuitContact` keyword is specified in the Math section or only the Poisson equation is solved, the Contact equation should be added explicitly in the Coupled statement to account for conductivity in metals.

To switch off the metal conductivity, specify the following keyword in the Math section:

```
Math{ -MetalConductivity }
```

The thermal conductivity of metals is simulated according to the thermodynamic model with the Joule heat:

$$c_L \frac{\partial T}{\partial t} - \nabla \cdot \kappa \nabla T = -\nabla \psi_M \cdot \vec{j}_M \quad (72)$$

where κ is the lumped electron–hole–lattice thermal conductivity (see [Thermoelectric Power \(TEP\) on page 649](#)). If it is switched off, the right-hand side of [Eq. 72](#) vanishes. However, a heat flow in metals is simulated.

Conductive Insulators

Leakage currents in dielectrics can be modeled by allowing dielectrics to have conductive properties. The conductive insulator (leaky insulator) model enables the computation of leaking currents through such dielectrics. A conductive insulator can have nonzero conductivity (metal-like property), typically a low conductivity, besides the pure dielectric properties. When conductivity is zero, the conductive insulator becomes simply an insulator.

4: Introduction to Physics in Sentaurus Device

Transport Equations

In a conductive insulator, the electrostatic potential is computed using the Poisson (Laplace) equation based on its dielectric properties, similar to a pure dielectric. As in metals, conductivity is modeled by the Fermi potential (solving Eq. 68) and then the leakage current is computed using Eq. 67. Since the model does not allow net charge in the conductive insulator, the Poisson equation and the electrostatic potential remain unchanged by adding conductive properties to the insulator.

The equation for Fermi potential inside a conductive insulator is the one used for metals (see Eq. 68), where σ is now the conductivity of the conductive insulator and Φ_{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, the default condition (Ohmic-like) can be written as:

$$\begin{aligned}\vec{J}_{CI} \cdot \hat{n} &= (\alpha \vec{J}_n + (1 - \alpha) \vec{J}_p) \cdot \hat{n} \\ \Phi_{CI} &= \phi - \phi_0\end{aligned}\quad (73)$$

where Φ_{CI} and \vec{J}_{CI} are the Fermi potential and current density on the interface, ϕ is the electrostatic potential (the solution of the Poisson equation in insulator and semiconductor regions), ϕ_0 is the equilibrium electrostatic potential (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, and α is the fraction ($0 \leq \alpha \leq 1$) of the current density of the conductive insulator going to the electron current density.

It is also possible to have a thermionic-like condition. In this case, the boundary condition becomes:

$$\begin{aligned}\vec{J}_{CI} \cdot \hat{n} &= (\vec{J}_{n, \text{th}} + \vec{J}_{p, \text{th}}) \cdot \hat{n} \\ \vec{J}_{n, \text{th}} &= aq \left[v_n(T_n)n - v_n(T) \left(\frac{T}{T_n} \right)^{1.5} N_C \exp \left(\frac{E_F^{\text{CI}} - E_C}{kT} \right) \right] \\ \vec{J}_{p, \text{th}} &= aq \left[v_p(T_p)p - v_p(T) \left(\frac{T}{T_p} \right)^{1.5} N_V \exp \left(\frac{E_V - E_F^{\text{CI}}}{kT} \right) \right]\end{aligned}\quad (74)$$

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; E_F^{CI} , E_C , and E_V are the conductive insulator Fermi-level energy, conduction band energy, and valence band energy,

respectively; and a is a constant set through the parameter file, which has the default value of 2.

When the semiconductor region has the charge boundary condition specified (it is a semiconductor floating gate), the boundary condition becomes:

$$\vec{J}_{\text{CI}} \cdot \hat{n} = -q\sigma\nabla\Phi \quad (75)$$

where Φ 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_{CI} is the total current at the conductive insulator–floating gate interface, and Δt is the current time step.

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 interface condition in [Eq. 73](#) and the resistivity of the conductive insulator specified in the parameter file. As can be noted in [Eq. 73](#), Sentaurus Device allows you to control which fraction of the current at the interface goes to the electron current and which fraction goes to the hole current through the parameter α , which is also specified in the parameter file.

There is also the possibility of having thermionic emission at the semiconductor–conductive insulator interface. In this case, conduction is limited by both interface properties and the bulk resistivity of the conductive insulator. Thermionic emission at the interface is modeled by [Eq. 74](#) with parameter a specified in the parameter file.

By using a conductive insulator layer between a semiconductor floating-gate and a semiconductor region, leakage from a semiconductor floating-gate can be emulated. This is a direct application of the boundary condition defined by [Eq. 75](#). The activation 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
    ...
}
```

4: Introduction to Physics in Sentaurus Device

Transport Equations

```
Solve {
    ...
    Coupled {Poisson Electron Hole Contact CondInsulator}
    ...
}
```

The bulk resistivity-limited case is the default, where conductive properties of conductive insulators and the fraction of the current passing to or from the semiconductor from or to the conductive insulator (parameter α in Eq. 73) can be specified in the parameter file:

```
Material = "Si3N4" {
    Resistivity {
        * Resist(T) = Resist0 * (1 + TempCoef * (T-273))
        Resist0 = 3.0e9 # [Ohm*cm]
        TempCoef = 43.0e-4 # [1/K]
    }
}

MaterialInterface = "Si3N4/AlGaN" {
    CondInsCurr {
        * ConductiveInsulator/Semiconductor current parameter
        * jc = curw*je + (1-curw)*jh
        * where:
        *     jc is the conduction current density
        *     je is the electron current density
        *     jh is the hole current density
        *     ecurw is fraction of current converted to je
        *     hcurw is fraction of current converted to jh
        curw = 1.0 # [1]
    }
}
```

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 Eq. 74) is specified in the parameter file:

```
MaterialInterface = "Si3N4/AlGaN" {
    * interface
    ThermionicEmission {
        A = 2, 2
    }
}
```

For nonthermionic interfaces (the bulk resistivity-limited case), an equilibrium solution is needed internally to solve the `CondInsulator` equation. In this case, the equilibrium solution is automatically computed with the default equation parameters. These parameters may not always be appropriate, especially in the case of III-V semiconductor devices where a large number of iterations is needed for convergence at the beginning of the simulation.

As an option, Sentaurus Device allows you to control the equilibrium solution parameters by specifying the keyword `EquilibriumSolution` in the `Math` section and re-defining the required parameters:

```
Math {
    ...
    EquilibriumSolution(... Iterations=100 ...)
    ...
}
```

When thermionic emission is used at semiconductor-conductive insulator interfaces, no equilibrium solution is needed for the model.

Current Potential

The total current density:

$$\vec{J} = \vec{J}_n + \vec{J}_p + \vec{J}_D \quad (76)$$

satisfies the conservation law:

$$\nabla \cdot \vec{J} = 0 \quad (77)$$

Consequently, \vec{J} can be written as the curl of a vector potential \vec{W} :

$$\vec{J} = \nabla \times \vec{W} \quad (78)$$

4: Introduction to Physics in Sentaurus Device

Transport Equations

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:

$$\vec{J} = \begin{bmatrix} \frac{\partial W}{\partial y} \\ \frac{\partial W}{\partial x} \end{bmatrix} \quad (79)$$

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 [23].

To visualize the results, add the keyword `CurrentPotential` to the `Plot` section:

```
Plot {
    CurrentPotential
}
```

[Figure 26](#) displays plots of the total current density and current potential for a simple square device.

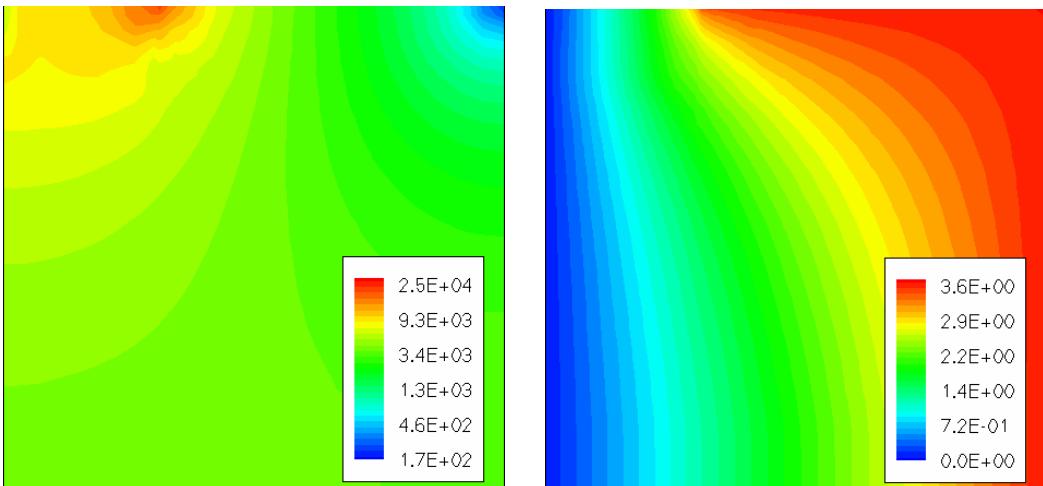


Figure 26 Total current density [Acm^{-1}] (*left*) and current potential [Acm^{-1}] (*right*)

Quasi-Fermi Potential

Electron and hole densities can be recomputed from the electron and hole quasi-Fermi potentials, and vice versa, using well-known formulas. If Boltzmann statistics is assumed, these formulas read:

$$n = N_C \exp\left(\frac{E_{F,n} - E_C}{kT}\right) \quad (80)$$

$$p = N_V \exp\left(\frac{E_V - E_{F,p}}{kT}\right) \quad (81)$$

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, and Φ_n and Φ_p are electron and hole quasi-Fermi potentials, respectively. E_C and E_V are conduction and valence band edges, defined as:

$$E_C = -\chi - q(\phi - \phi_{ref}) \quad (82)$$

$$E_V = -\chi - E_{g,eff} - q(\phi - \phi_{ref}) \quad (83)$$

where χ denotes the electron affinity and $E_{g,eff}$ the effective band gap.

Electrostatic potential ϕ is computed from an arbitrarily defined reference potential ϕ_{ref} . For pure materials and, in particular, for silicon, the standard approach is to set the reference potential equal to the Fermi potential of an intrinsic semiconductor. Then, Eq. 80 and Eq. 81 can be written as:

$$n = n_{i,eff} \cdot \exp\left(\frac{-q(\Phi_n - \phi)}{kT}\right) \quad (84)$$

$$p = n_{i,eff} \cdot \exp\left(\frac{q(\Phi_p - \phi)}{kT}\right) \quad (85)$$

where $n_{i,eff}$ is the effective intrinsic density.

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 Eq. 80 or Eq. 81. 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.

4: Introduction to Physics in Sentaurus Device

Fermi Statistics

NOTE In many cases if avalanche generation is important, the one carrier approximation cannot be applied even for unipolar devices.

Fermi Statistics

For the equations presented in the previous sections, Boltzmann statistics was assumed for electrons and holes. In a more general consideration, the Fermi (also called Fermi–Dirac) distribution function 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, Eq. 80 and for Eq. 81 are replaced by:

$$n = N_C F_{1/2} \left(\frac{E_{F,n} - E_C}{kT} \right) \quad (86)$$

$$p = N_V F_{1/2} \left(\frac{E_V - E_{F,p}}{kT} \right) \quad (87)$$

where $F_{1/2}$ is the Fermi integral of order 1/2. Sentaurus Device uses the Joyce–Dixon approximation [24] to compute the Fermi integral and its inverse function.

In place of Eq. 84 and Eq. 85, the following equations are valid with Fermi statistics:

$$n = n_{i,\text{eff}} \cdot \gamma_n \exp \left(\frac{-q(\Phi_n - \phi)}{kT} \right) \quad (88)$$

$$p = n_{i,\text{eff}} \cdot \gamma_p \exp \left(\frac{q(\Phi_p - \phi)}{kT} \right) \quad (89)$$

where γ_n and γ_p are the functions of η_n and η_p :

$$\gamma_n = \frac{n}{N_C} \exp(-\eta_n) \quad (90)$$

$$\gamma_p = \frac{p}{N_V} \exp(-\eta_p) \quad (91)$$

$$\eta_n = \frac{E_{F,n} - E_C}{kT} \quad (92)$$

$$\eta_p = \frac{E_V - E_{F,p}}{kT} \quad (93)$$

In the case of Fermi statistics, [Eq. 36, p. 185](#) and [Eq. 41, p. 186](#) for the electron current density and electron energy flux density are replaced, respectively, by:

$$\vec{J}_n = \mu_n \left(qn\nabla E_C + kT_n \nabla n - nkT_n \nabla (\ln \gamma_n) + \lambda_n f_n^{td} kn \nabla T_n - 1.5nkT_n \nabla \ln m_n \right) \quad (94)$$

$$\vec{S}_n = -\frac{5}{2} r_n \lambda_n \left(\frac{kT_n}{q} \vec{J}_n + f_n^{\text{hf}} \hat{\kappa}_n \nabla T_n \right) \quad (95)$$

where:

$$\lambda_n = \frac{F_{1/2}(\eta_n)}{F_{-1/2}(\eta_n)} \quad (96)$$

Similar changes are made in [Eq. 37, p. 185](#) and [Eq. 42, p. 186](#) for the hole current density and hole energy flux density, respectively.

Using Fermi Statistics

To activate Fermi statistics, the keyword `Fermi` must be specified in the global `Physics` section:

```
Physics {
    ...
    Fermi
}
```

NOTE Fermi statistics can be activated only for the whole device. Region-specific or material-specific activation is not possible, and the keyword `Fermi` is ignored in any `Physics` section other than the general one. For heterostructures, the keyword `Fermi` must appear in the general `Physics` section.

Multivalley Statistics

[Eq. 86, p. 202](#) and [Eq. 87, p. 202](#) 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.

4: Introduction to Physics in Sentaurus Device

Multivalley Statistics

With the parabolic band assumption and Fermi–Dirac distribution function applied to each valley, the multivalley electron and hole concentrations are represented as follows:

$$n = N_C F_{MV,n} \left(\frac{E_{F,n} - E_C}{kT} \right) = N_C \sum_{i=1}^{N_n} g_n^i F_{1/2} \left(\frac{E_{F,n} - E_C - \Delta E_n^i}{kT} \right) \quad (97)$$

$$p = N_V F_{MV,p} \left(\frac{E_V - E_{F,p}}{kT} \right) = N_V \sum_{i=1}^{N_p} g_p^i F_{1/2} \left(\frac{\Delta E_p^i + E_V - E_{F,p}}{kT} \right) \quad (98)$$

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, and ΔE_n^i and ΔE_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 [Eq. 86, p. 202](#) and [Eq. 87, p. 202](#).

Similarly to the Fermi statistics, the inverse functions of $F_{MV,n}$ and $F_{MV,p}$ are used to define the variables γ_n and γ_p that bring an additional drift term with $\nabla(\ln\gamma)$ as it is expressed in [Eq. 94, p. 203](#):

$$\gamma_n = \frac{n}{N_C} \exp \left(-F_{MV,n}^{-1} \left(\frac{n}{N_C} \right) \right) \quad (99)$$

$$\gamma_p = \frac{p}{N_V} \exp \left(-F_{MV,p}^{-1} \left(\frac{p}{N_V} \right) \right) \quad (100)$$

To compute the inverse functions $F_{MV,n}^{-1} \left(\frac{n}{N_C} \right)$ and $F_{MV,p}^{-1} \left(\frac{p}{N_V} \right)$, an internal Newton solver is used.

Using Multivalley Statistics

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.

NOTE Together with the `Fermi` keyword (see [Fermi Statistics on page 202](#)), [Eq. 97](#) and [Eq. 98](#) are used. 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.

NOTE For stress simulations (with the keyword `Piezo` in the `Physics` section), all multivalley model parameters (N_n , N_p , g_n^i , g_p^i , ΔE_n^i , and ΔE_p^i) are defined by stress models (see [Multivalley Band Structure on page 605](#)), and user control of the parameters is possible only through these stress models.

For nonstress simulations, the 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 silicon), Sentaurus Device defines three electron valleys and two hole valleys as follows:

```
MultiValley{
    eValley(0.914, 0.196, 0.196, 0, 2, 0.5) #[1,1,1,eV,1,eV^-1]
    eValley(0.196, 0.914, 0.196, 0, 2, 0.5) #[1,1,1,eV,1,eV^-1]
    eValley(0.196, 0.196, 0.914, 0, 2, 0.5) #[1,1,1,eV,1,eV^-1]
    hValley(0.16, 0.16, 0.16, 0, 1, 0)      #[1,1,1,eV,1,eV^-1]
    hValley(0.49, 0.49, 0.49, 0, 1, 0)      #[1,1,1,eV,1,eV^-1]
}
```

A general format for each valley is $(e/h)\text{Valley}(m_{100}, m_{010}, m_{001}, \Delta E, d, \alpha)$, where:

- m_{100} , m_{010} , and m_{001} are the effective DOS masses along the crystal axis.
- ΔE is the valley energy shift with respect to the band edge, which defines ΔE_n^i and ΔE_p^i in [Eq. 97](#) and [Eq. 98](#).
- d is the degeneracy.
- α is the nonparabolicity parameter (for future use, that is, it is not used in the current implementation).

The model parameters g_n^i and g_p^i are defined by the effective DOS masses as follows:

$$g_{n,p}^i = \frac{d^i (m_{100}^i m_{010}^i m_{001}^i)^{0.5}}{\sum_{i=1}^N d^k (m_{100}^k m_{010}^k m_{001}^k)^{0.5}} \quad (101)$$

If all valley energy shifts equal zero, [Eq. 101](#) provides the single-valley condition where, as usual, only the effective DOS and band edge define the semiconductor band structure and carrier concentration.

Boundary Conditions

Electrical Boundary Conditions

Ohmic Contacts

Charge neutrality and equilibrium are assumed at the electrodes for Ohmic contacts:

$$n_0 - p_0 = N_D - N_A \quad (102)$$

$$n_0 p_0 = n_{i,eff}^2 \quad (103)$$

In the case of Boltzmann statistics, this system of equations is easily solved:

$$\phi = \phi_F + \frac{kT}{q} \operatorname{asinh} \left(\frac{N_D - N_A}{2n_{i,eff}} \right) \quad (104)$$

$$n_0 = \sqrt{\frac{(N_D - N_A)^2}{4} + n_{i,eff}^2} + \frac{N_D - N_A}{2}, \quad p_0 = \sqrt{\frac{(N_D - N_A)^2}{4} + n_{i,eff}^2} - \frac{N_D - N_A}{2} \quad (105)$$

where n_0, p_0 are the electron and hole equilibrium concentrations, and ϕ_F is the Fermi potential at the contact that is equal to the applied voltage if it is not a resistive contact (see [Resistive Contacts on page 210](#)). 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:

$$\vec{J}_n \cdot \hat{n} = qv_n(n - n_0) \quad \vec{J}_p \cdot \hat{n} = -qv_p(p - p_0) \quad (106)$$

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 This specification may lead to discontinuous electrostatic potential along the electrode if the electrode is in contact with both semiconductor and insulator regions, because it will assign zero electrostatic potential (with `Voltage=0`) for the insulator nodes. To have a realistic potential distribution, use the workfunction or material specifications described in the next section, [Gate Contacts](#). For example, in the case of deposited polysilicon, the following statement with a proper doping concentration in the poly can be useful:

`Material="PolySi" (N=1e20)`

NOTE If the values of the electron and hole recombination velocities are equal to zero ($v_n = 0$, $v_p = 0$), then $\vec{J}_n \cdot \hat{n} = 0$, $\vec{J}_p \cdot \hat{n} = 0$. This is a reflective (or ideal Neumann) boundary condition for current equations, that is, there is an electrode where only the potential is defined (which is useful for SOI devices).

Gate Contacts

For gate contacts, the electrostatic potential is taken as:

$$\phi = \phi_F - \Phi_{MS} \quad (107)$$

where ϕ_F is the Fermi potential at the contact that is equal to the applied voltage if it is not a resistive contact (see [Resistive Contacts on page 210](#)), and Φ_{MS} is the workfunction difference between the metal and an intrinsic reference semiconductor.

Φ_{MS} can be specified in the `Electrode` section, either directly with `Barrier` or indirectly by specifying the metal workfunction with `WorkFunction`:

```
Electrode { ...
  { Name="Gate" Voltage=0 Barrier = 0.55 }
}
Electrode { ...
  { Name="Gate" Voltage=0 WorkFunction = 5 }
}
```

A third possibility is to use a material name to define the electrode workfunction, for example:

```
Electrode { ...
  { Name="Gate" Voltage=0 Material="Gold" }
}
```

Then, the value for the workfunction is obtained from the parameter file:

```
Material = "Gold" {
  Bandgap { WorkFunction = 5 }
}
```

4: Introduction to Physics in Sentaurus Device

Boundary Conditions

NOTE If Material or WorkFunction is specified for an Ohmic contact that is in contact with both the semiconductor and insulator, the electrostatic potential will be equal to the built-in potential at semiconductor nodes, but for insulator nodes, it will correspond to [Eq. 107](#).

If you need to emulate a poly-semiconductor 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 the standard expression $(kT/q)\ln(N/n_i)$.

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 [\[25\]](#) and [\[26\]](#). In this section, a typical model for Schottky contacts [\[27\]](#) is considered. The following boundary conditions hold:

$$\phi = \phi_F - \Phi_B + \frac{kT}{q} \ln\left(\frac{N_C}{n_{i,eff}}\right) \quad (108)$$

$$\vec{J}_n \cdot \hat{n} = qv_n(n - n_0^B) \quad \vec{J}_p \cdot \hat{n} = -qv_p(p - p_0^B) \quad (109)$$

$$n_0^B = N_C \exp\left(\frac{-q\Phi_B}{kT}\right) \quad p_0^B = N_V \exp\left(\frac{-E_{g,eff} + q\Phi_B}{kT}\right) \quad (110)$$

where ϕ_F is the Fermi potential at the contact that is equal to an applied voltage V_{applied} if it is not a resistive contact (see [Resistive Contacts on page 210](#)), Φ_B is the barrier height (the difference between the metal workfunction and the electron affinity of the semiconductor), v_n and v_p are the thermionic emission velocities, and 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 $2.573 \times 10^6 \text{ cm/s}$ and $1.93 \times 10^6 \text{ cm/s}$, respectively.

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 can produce wrong results if the Schottky contact is connected to several different semiconductors. In such cases, use the specification WorkFunction instead of Barrier. See [Gate Contacts on page 207](#) for information about gate contacts for the specification of metal materials.

Barrier Lowering at Schottky Contacts

The barrier lowering model can be applied to a Schottky contact. This model can account for different physical mechanisms. The most important one is the image force [27], but it can also model tunneling and dipole effects.

The following expression is used to compute the value of the barrier lowering:

$$\Delta\Phi_B(F) = a_1 \left[\left(\frac{F}{F_0} \right)^{p_1} - \left(\frac{F_{eq}}{F_0} \right)^{p_{1,eq}} \right] + a_2 \left[\left(\frac{F}{F_0} \right)^{p_2} - \left(\frac{F_{eq}}{F_0} \right)^{p_{2,eq}} \right] \quad (111)$$

where $F_0 = 1 \text{ Vcm}^{-1}$, F_{eq} is the equilibrium electric field used to obtain $\Delta\Phi_B = 0$ at the equilibrium, and a_1 , a_2 , p_1 , $p_{1,eq}$, p_2 , and $p_{2,eq}$ are model coefficients that can be specified in the parameter file of Sentaurus Device. Their default values are $a_1 = 2.6 \times 10^{-4} \text{ eV}$, $p_1 = p_{1,eq} = 0.5$, $a_2 = 0 \text{ eV}$, and $p_2 = p_{2,eq} = 1$.

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 major carriers. The barrier lowering also affects the equilibrium concentration of electrons n_0^B and holes p_0^B corresponding to its formula [Eq. 110](#).

To activate the barrier lowering model, specify `BarrierLowering` in the electrode-specific `Physics` section:

```
Physics(Electrode = "Gate") { BarrierLowering }
```

To specify parameters of the model, create the following parameter set:

```
Electrode = "Gate" {
  BarrierLowering {
    a1 = a1 [eV]
    p1 = p1 [1]
```

4: Introduction to Physics in Sentaurus Device

Boundary Conditions

```

p1_eq = p1, eq [1]
a2 = a2 [eV]
p2 = p2 [1]
p2_eq = p2, eq
}
}

```

Resistive Contacts

If V_{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, see units in [Table 146 on page 1110](#)), there is an additional equation to compute ϕ_F in [Eq. 104](#), [Eq. 107](#), and [Eq. 109](#).

For a distributed resistance, ϕ_F is different for each mesh vertex of the contact and is computed as a solution of the following equation, which is solved for each contact vertex self-consistently with the system of all equations:

$$\hat{n} \cdot [\vec{J}_p(\phi_F) + \vec{J}_n(\phi_F) + \vec{J}_D(\phi_F)] = \frac{(V_{\text{applied}} - \phi_F)}{R_d} \quad (112)$$

For a resistor, ϕ_F is a constant over the contact and only one of the following equations per contact is solved self-consistently with the system of all equations:

$$\int_s \hat{n} \cdot [\vec{J}_p(\phi_F) + \vec{J}_n(\phi_F) + \vec{J}_D(\phi_F)] ds = \frac{(V_{\text{applied}} - \phi_F)}{R} \quad (113)$$

where s is a contact area used in a Sentaurus Device simulation to compute a total current through the contact.

NOTE In 2D simulations, R must be specified in units of $\Omega \mu\text{m}$. In 3D, R must be specified in units of Ω (see [Table 146 on page 1110](#)).

From emulating a behavior of a Schottky contact [28], Schottky contact resistivity at zero bias was derived and a doping-dependent resistivity model of such contacts was obtained. This model is applied to Ohmic contacts and is activated by the keyword DistResist = SchottkyResist in the Electrode section.

The model is expressed as:

$$\begin{aligned}
 R_d &= R_\infty \frac{300\text{ K}}{T_0} \exp\left(\frac{q\Phi_B}{E_0}\right) \\
 E_0 &= E_{00} \coth\left(\frac{E_{00}}{kT_0}\right) \\
 E_{00} &= \frac{qh}{4\pi} \sqrt{\frac{|N_{D,0} - N_{A,0}|}{\epsilon_s m_t}}
 \end{aligned} \tag{114}$$

where:

- Φ_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_∞ is the Schottky resistance for an infinite doping concentration at the contact (or zero Schottky barrier).
- ϵ_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 159](#) on [page 1125](#)).

The parameters of the model can be specified in an electrode-specific parameter set. The allowed syntax and recommended default values of these parameters [28] are:

```

SchottkyResistance {
    Rinf = 2.4000e-09 , 5.2000e-09 # [Ohm*cm^2]
    PhiB = 0.6 , 0.51      # [eV]
    mt = 0.19 , 0.16      # [1]
}

```

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.

Floating Metal Gates

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 }
}

```

4: Introduction to Physics in Sentaurus Device

Boundary Conditions

In the case of a floating metal gate (that is defined only by a contact with no associated semiconductor region), the electrostatic potential is determined by solving the Poisson equation with the following charge boundary condition:

$$\int \epsilon \hat{n} \cdot \nabla \phi dS = Q \quad (115)$$

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 $\phi_{FC} = \phi_S$ on the floating contact surface is assumed to be constant.

An additional floating or control gate capacitance 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 floating gate capacitance can be specified in the Electrode section using the keyword FGcap.

For example, if we have ContGate and FloatGate electrodes, additional ContGate/FloatGate capacitance is specified for the floating electrode:

```
Electrode {
    { name="ContGate" voltage=10 }
    { name="FloatGate" charge=0 FGcap=(value=3e-15 name="ContGate") }
}
```

where value is the capacitance value between FloatGate and the electrode ContGate. For the 1D case, the capacitance units are $F/\mu\text{m}^2$; for the 2D case, $F/\mu\text{m}$; and for the 3D case, F .

If the floating-gate capacitance is specified, Eq. 115 changes to:

$$\int \epsilon \hat{n} \cdot \nabla \phi dS + C_{FG}(\phi_{CG} - \phi_{FG} + \phi_B) = Q \quad (116)$$

where C_{FG} is the specified floating-gate capacitance, ϕ_{FG} is the floating-gate potential, ϕ_{CG} is the potential on the ‘ControlGate’ electrode (specified in the FGcap statement), and ϕ_B is a barrier specified in the Electrode section of the floating gate, which can be used as a fitting parameter (default value is zero).

NOTE If the ‘ControlGate’ electrode is a metal, then a value of ϕ_{CG} is defined by the applied voltage and the metal barrier/workfunction (see [Gate Contacts on page 207](#)). However, if the electrode is in contact with the semiconductor–poly region, then ϕ_{CG} is equal to the applied voltage with an averaged built-in potential on the electrode.

The `FloatGate` electrodes can have several capacitance values for different electrodes, 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 17 on page 511](#) and [Chapter 18 on page 531](#), respectively.

Floating Semiconductor Gates

Sentaurus Device can simulate floating semiconductor gates. Within a floating semiconductor (as opposed to metal) gate, Sentaurus Device solves the Poisson equation with the following charge boundary condition:

$$q \int_{\text{FG}} [q(p - n + N_D - N_A) + \rho_{\text{trap}}] dV = Q_{\text{FG}} \quad (117)$$

where Q_{FG} denotes the total charge on the floating gate and ρ_{trap} is the charge density contributed by traps and fixed charges (see [Chapter 10 on page 349](#)). The integral is calculated over all nodes of the floating gate region [29].

The charge Q_{FG} is a boundary condition that must be specified in a Sentaurus Device `Electrode` statement in exactly the same way as for a floating metal gate:

```
Electrode {
  { name="floating_gate" Charge=1e-14 }
}
```

Sentaurus Device automatically identifies a floating semiconductor gate 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 gate. A small contact is sufficient if the gate material has the same doping type throughout. However, if both n-type and p-type volumes exist in the gate region, separate contacts are required for each.

It is assumed that no current flows within the floating gate. Therefore, the quasi-Fermi potential for electrons and holes must be identical and constant within the floating gate:

$$\Phi_n = \Phi_p = \Phi \quad (118)$$

4: Introduction to Physics in Sentaurus Device

Boundary Conditions

Therefore, the electron and hole densities n and p are functions of the electrostatic potential ϕ and the quasi-Fermi potential Φ as discussed in [Quasi-Fermi Potential on page 201](#) and [Fermi Statistics on page 202](#). Sentaurus Device does not solve the electron and hole continuity equations within a floating gate.

For a steady state simulation, the charge of the floating gate must be specified as a boundary condition in the `Electrode` section. However, 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} }
```

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 17 on page 511](#) and [Chapter 18 on page 531](#). For hot-carrier injection models, see [Chapter 18 on page 531](#).

The floating-gate capacitance can be specified in the `Electrode` statement in exactly the same way as for the floating metal gate (see [Floating Metal Gates on page 211](#)). The only difference is that the potential on the semiconductor floating gate may not be a constant (compared to the floating metal gate) due to a depletion and change in the doping. Therefore, defining a constant capacitance may not be strictly correct, and it gives only approximate results. Sentaurus Device uses the semiconductor floating-gate Fermi level to compute the charge associated with the floating-gate capacitance and solves an equilibrium problem (zero voltages and zero charges) at the beginning to find a reference Fermi level.

For plotting purposes, floating semiconductor gates are handled differently than other electrodes. Instead of the voltage, the value of the quasi-Fermi potential Φ is displayed.

Boundaries Without Contacts

All other boundaries are treated with reflective (or ideal Neumann) boundary conditions:

$$\vec{F} \cdot \hat{n} = 0 \quad (119)$$

$$\vec{J}_n \cdot \hat{n} = 0 \quad \vec{J}_p \cdot \hat{n} = 0 \quad (120)$$

Thermal Boundary Conditions for Thermodynamic Model

For the solution of [Eq. 32, p. 180](#), correct thermal boundary conditions must be applied. Wachutka [30] is followed, but the difference in thermo-powers between the semiconductor and metal at Ohmic contacts is neglected. For free, thermally insulating surfaces:

$$\kappa \hat{n} \cdot \nabla T = 0 \quad (121)$$

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:

$$\kappa \hat{n} \cdot \nabla T = \frac{T_{\text{ext}} - T}{R_{\text{th}}} \quad (122)$$

where R_{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:

$$T = T_{\text{ext}} \quad (123)$$

Dirichlet and nonhomogeneous Neumann boundary conditions (R_{th}) are specified in the Thermode section (see [Thermode Section on page 53](#)).

When the lattice temperature equation ([Eq. 32, p. 180](#)) is solved, the lattice temperature and lattice heat flux at thermodes are automatically added to the .plt file in the corresponding thermode section. The unit for heat flux is W for 3D, W/ μm for 2D, and W/ μm^2 for 1D.

Thermal Boundary Conditions for Hydrodynamic Model

Boundary conditions for the lattice temperature T in the hydrodynamic model are specified in the same way as thermal boundary conditions for the thermodynamic model (see [Thermal Boundary Conditions for Thermodynamic Model on page 215](#)). 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:

$$\kappa_n \hat{n} \cdot \nabla T_n = \kappa_p \hat{n} \cdot \nabla T_p = 0 \quad (124)$$

When the hydrodynamic model is used, the lattice temperature, lattice heat flux, and carrier heat fluxes at thermodes are automatically added to the .plt file in the corresponding thermode section. The unit for heat fluxes is W for 3D, W/ μm for 2D, and W/ μm^2 for 1D.

Total Thermal Resistance

The thermal behavior of a semiconductor device is described in industry-standard nomenclature by the thermal resistance R_{ja} from the ‘junction’ or electrically active, heat-producing area of the device, to the ‘ambient’ or environment. The maximum device temperature at given operating conditions depends on R_{ja} . For self-heating simulations, R_{ja} can be broken down into an *internal* component that is taken into account through the thermal conductivity κ in [Eq. 29](#), and an *external* component that is defined by R_{th} and provided by you.

The determination of R_{th} can be crucial for accurate thermal simulations since the bulk of the thermal resistance usually lies external to the usual electrical simulation domain. One approach is to simulate as much as possible of the thermal environment (die, heat sink, packaging) and estimate a reasonable external thermal resistance R_{th} based on results of thermal measurements [\[31\]–\[34\]](#) and thermal simulations [\[35\]–\[38\]](#) of semiconductor devices and packages.

Including as much as possible of the thermal environment in the simulation domain has two important advantages:

- As a larger fraction of the total thermal resistance R_{ja} is accurately accounted for in the simulation domain, a smaller fraction is left for estimation through the choice of R_{th} .
- Nonuniform temperature distributions in the die, heat sink, and packaging are allowed. This permits the simulation of a realistic temperature distribution in the electrically active area for more accurate determination of self-heating effects.

The addition of large portions of the heat sink and packaging is not a problem with regard to the total number of mesh points, despite the juxtaposition of fine mesh requirements in the electrically active area with large, coarse mesh areas in the die, heat sink, and packaging. For a typical 2D mesh of approximately 1000 mesh points, such an extension of the thermal simulation domain usually adds no more than 10% to the total number of mesh points.

Estimating Thermal Resistance

To estimate a value for R_{th} , the following must be considered:

- Bulk resistance

The bulk thermal resistance can be calculated per unit area given the thermal conductivity κ and the length l of the material through which heat flows by the relation $R_{\text{th,bulk}} = l/\kappa$.

- Interface resistances

Apart from the effects of bulk thermal resistance, interfaces between materials act as barriers to heat flow. Interface thermal resistances can result from lattice mismatch, surface roughness [39], or air voids (for example, in solder joints) [40] at crystal–crystal, crystal–metal, and metal–metal interfaces. The effect of interface thermal resistances on R_{th} can be large [41].

- Radiation/convection

The effects of heat radiation and convection can be taken into account using the linear cooling law represented by Eq. 122, [42]. The value of R_{th} is appropriately reduced when radiation or convection has an important role in cooling [43]–[45].

- Thermal spreading effects

Since the actual heat flow path in the physical device is in 3D, the value of the external resistance must be estimated taking into account the 3D geometry [46] by using an approach similar to one documented [47].

The values of R_{th} used in self-heating simulations are usually in the range $0.1 \leq R_{\text{th}} \leq 1.0 \text{ KW}^{-1}\text{cm}^2$.

Periodic Boundary Conditions

Periodic boundary conditions (PBCs) can be activated in Sentaurus Device for all supported equations (see Transport Equations on page 177). For ‘left’ and ‘right’ boundaries, it provides the following conditions for a selected equation:

$$\begin{aligned} v_{\text{left}} &= v_{\text{right}} \\ \rightarrow &\quad \rightarrow \\ \Phi_{\text{left}} &= \Phi_{\text{right}} \end{aligned} \tag{125}$$

where v is a solution variable of the selected equation (for example, the electrostatic potential ϕ of the Poisson equation) and Φ is a flux that is defined in $\nabla \cdot \Phi$ of the equation (see Table 23 on page 218).

4: Introduction to Physics in Sentaurus Device

Boundary Conditions

Table 23 Solution variables and fluxes used in periodic boundary conditions

v	$\vec{\Phi}$	Description
ϕ	$\varepsilon \nabla \phi$	Poisson Equation and Continuity Equations on page 178
n	\vec{J}_n	Drift-Diffusion Model on page 179
p	\vec{J}_p	
T_n	\vec{S}_n	Hydrodynamic Transport Model on page 184
T_p	\vec{S}_p	
T	$\kappa \nabla T$	Thermodynamic Transport Model on page 179

The PBCs can be activated by the keyword `PeriodicBC(<options>)` in the Math section where `<options>` provides a selection of the equations and boundaries. Multiple specifications of `<options>` are possible to select several equations: `PeriodicBC((<options1>) ... (<optionsN>))`.

[Table 149 on page 1112](#) gives a complete list of possible options.

NOTE To use PBCs, the materials, doping concentration, and mesh points on both sides must coincide.

An example of specifying the periodic boundary conditions 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))
    )
}
```

Here, the first option applies PBCs to all equations in the direction of the x-axis and for the coordinates $-1.0\text{ }\mu\text{m}$ and $2.0\text{ }\mu\text{m}$. The next two options specify PBCs for electron and hole continuity equations in the y-direction and for the device side coordinates.

Starting Solution or ‘Initial Guess’

The starting solution or ‘initial guess’ is determined in Sentaurus Device using a predetermined algorithm. An initial guess of each variable (at each mesh point) in a device required by the solution method consists of the following values:

- Electrostatic potential (ϕ)
- Quasi-Fermi potentials for electrons and holes (Φ_n and Φ_p) for the electrical case
- Lattice temperature (T) for the thermodynamic case
- Electron and hole temperatures (T_n and T_p) for the hydrodynamic case

Electrostatic Potential and Quasi-Fermi Potentials in Doping Wells

To determine an initial guess for the electrostatic potential and quasi-Fermi potentials, the 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 non-semiconductor elements or by vacuum (a doping well region may 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 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).

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:

$$\Phi_p = k_{\text{float}} V_{\max} + (1 - k_{\text{float}}) V_{\min} \quad (126)$$

$$\Phi_n = (1 - k_{\text{float}}) V_{\max} + k_{\text{float}} V_{\min} \quad (127)$$

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_{float} is an adjustable parameter with a default value of 0. To change the value of k_{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} described above.

4: Introduction to Physics in Sentaurus Device

Starting Solution or 'Initial Guess'

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.

Regionwise Specification of Initial Quasi-Fermi Potentials

You can set initial quasi-Fermi potentials regionwise. This is especially useful in CCD simulations, where a CCD cell or region must be initially in a certain state (usually, deep depletion). By specifying a 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.

Thermodynamic and Hydrodynamic Simulations

By default, the lattice temperature is set to 300K 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. The electron and hole temperatures are set by initially making an estimation to the lattice temperature.

Save File Overrides the Initial Guess

If created from a previous solution, the save file (with extension `.sav`) can be loaded using the keyword `Load` in the `File` section. In this case, the values of the variables in this file overwrite the initial guess values.

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.
- [3] K. Kells, *General Electrothermal Semiconductor Device Simulation*, Series in Microelectronics, vol. 37, Konstanz, Germany: Hartung-Gorre, 1994.
- [4] R. Stratton, “Diffusion of Hot and Cold Electrons in Semiconductor Barriers,” *Physical Review*, vol. 126, no. 6, pp. 2002–2014, 1962.
- [5] K. Bløtekjær, “Transport Equations for Electrons in Two-Valley Semiconductors,” *IEEE Transactions on Electron Devices*, vol. ED-17, no. 1, pp. 38–47, 1970.
- [6] J. W. Roberts and S. G. Chamberlain, “Energy-Momentum Transport Model Suitable for Small Geometry Silicon Device Simulation,” *COMPEL*, vol. 9, no. 1, pp. 1–22, 1990.
- [7] 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.
- [8] A. Benvenuti *et al.*, “Analysis of output NDR in power AlGaAs/GaAs HBTs by means of a Thermal-Fully Hydrodynamic model,” in *International Semiconductor Device Research Symposium (ISDRS)*, vol. 2, Charlottesville, VA, USA, pp. 499–502, December 1993.
- [9] S. Szeto and R. Reif, “A Unified Electrothermal Hot-Carrier Transport Model for Silicon Bipolar Transistor Simulations,” *Solid-State Electronics*, vol. 32, no. 4, pp. 307–315, 1989.
- [10] A. Pierantoni *et al.*, “Three-Dimensional Implementation of a Unified Transport Model,” in *Simulation of Semiconductor Devices and Processes (SISDEP)*, vol. 5, Vienna, Austria, pp. 125–128, September 1993.
- [11] 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.
- [12] *ESPRIT-6075 Project (DESSIS), Period 1, May 27 – November 26, 1992*, Six-Month Report, University of Bologna, Bologna, Italy, 1992.
- [13] A. Bringer and G. Schön, “Extended moment equations for electron transport in semiconducting submicron structures,” *Journal of Applied Physics*, vol. 64, no. 5, pp. 2447–2455, 1988.

4: Introduction to Physics in Sentaurus Device

References

- [14] M. A. Stettler, M. A. Alam, and M. S. Lundstrom, “A Critical Examination of the Assumptions Underlying Macroscopic Transport Equations for Silicon Devices,” *IEEE Transactions on Electron Devices*, vol. 40, no. 4, pp. 733–740, 1993.
- [15] G. Baccarani and M. R. Wordeman, “An Investigation of Steady-State Velocity Overshoot in Silicon,” *Solid-State Electronics*, vol. 28, no. 4, pp. 407–416, 1985.
- [16] E. M. Azoff, “Semiclassical high-field transport equations for nonparabolic heterostructure degenerate semiconductors,” *Journal of Applied Physics*, vol. 64, no. 5, pp. 2439–2446, 1988.
- [17] M. Stecher *et al.*, “On the Influence of Thermal Diffusion and Heat Flux on Bipolar Device and Circuit Performance,” in *Simulation of Semiconductor Devices and Processes (SISDEP)*, vol. 5, Vienna, Austria, pp. 49–52, September 1993.
- [18] M. C. Vecchi and L. G. Reyna, “Generalized Energy Transport Models for Semiconductor Device Simulation,” *Solid-State Electronics*, vol. 37, no. 10, pp. 1705–1716, 1994.
- [19] Y. V. Apanovich *et al.*, “Steady-State and Transient Analysis of Submicron Devices Using Energy Balance and Simplified Hydrodynamic Models,” *IEEE Transactions on Computer-Aided Design*, vol. 13, no. 6, pp. 702–711, 1994.
- [20] D. Chen *et al.*, “An Improved Energy Transport Model Including Nonparabolicity and Non-Maxwellian Distribution Effects,” *IEEE Transactions on Electron Devices*, vol. 39, no. 1, pp. 26–28, 1992.
- [21] A. Benvenuti *et al.*, “Coupled Thermal-Fully Hydrodynamic Simulation of InP-based HBTs,” in *IEDM Technical Digest*, San Francisco, CA, USA, pp. 737–740, December 1992.
- [22] A. Benvenuti, G. Ghione, and C. U. Naldi, “Non-Stationary Transport HBT Modeling Under Non-Isothermal Conditions,” in *Simulation of Semiconductor Devices and Processes (SISDEP)*, vol. 5, Vienna, Austria, pp. 453–456, September 1993.
- [23] 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.
- [24] W. B. Joyce and R. W. Dixon, “Analytic approximations for the Fermi energy of an ideal Fermi gas,” *Applied Physics Letters*, vol. 31, no. 5, pp. 354–356, 1977.
- [25] 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.
- [26] A. Schenk, *Advanced Physical Models for Silicon Device Simulation*, Wien: Springer, 1998.
- [27] S. M. Sze, *Physics of Semiconductor Devices*, New York: John Wiley & Sons, 2nd ed., 1981.

- [28] 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.
- [29] 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.
- [30] 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.
- [31] J. A. Andrews, “Package Thermal Resistance Model: Dependency on Equipment Design,” *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, vol. 11, no. 4, pp. 528–537, 1988.
- [32] J. N. Sweet and W. T. Cooley, “Thermal Resistance Measurements and Finite Element Calculations for Ceramic Hermetic Packages,” in *Sixth Annual IEEE Semiconductor Thermal and Temperature Measurement Symposium (SEMI-THERM)*, Scottsdale, AZ, USA, pp. 10–16, February 1990.
- [33] T. Hopkins, C. Cognetti, and R. Tiziani, “Designing with Thermal Impedance,” in *Fourth Annual IEEE Semiconductor Thermal and Temperature Measurement Symposium (SEMI-THERM)*, San Diego, CA, USA, pp. 55–61, February 1988.
- [34] R. L. Kozarek, “Effect of Case Temperature Measurement Errors on the Junction-to-Case Thermal Resistance of a Ceramic PGA,” in *Seventh Annual IEEE Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*, Phoenix, AZ, USA, pp. 44–51, February 1991.
- [35] Y. J. Min, A. L. Palisoc, and C. C. Lee, “Transient Thermal Study of Semiconductor Devices,” *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, vol. 13, no. 4, pp. 980–988, 1990.
- [36] F. Curatelli and G. M. Bisio, “Characterization of the Thermal Behaviour in ICs,” *Solid-State Electronics*, vol. 34, no. 7, pp. 751–760, 1991.
- [37] G. N. Ellison, “TAMS: A Thermal Analyzer for Multilayer Structures,” *Electrosoft*, vol. 1, no. 2, pp. 85–97, 1990.
- [38] R. A. Tatara, “Thermal Modeling Previews Electronic Device Performance,” *PCIM*, pp. 9–21, October 1991.
- [39] S. Song and M. M. Yovanovich, “Relative Contact Pressure: Dependence on Surface Roughness and Vickers Microhardness,” *Journal of Thermophysics and Heat Transfer*, vol. 2, no. 1, pp. 43–47, 1988.
- [40] V. P. Deshwal *et al.*, “Optimum thickness determination of the electrode for large silicon power devices and its improved bonding with silicon wafer having N-doped substrate,” *Indian Journal of Technology*, vol. 29, no. 8, pp. 395–398, 1991.

4: Introduction to Physics in Sentaurus Device

References

- [41] W. S. Childres and G. P. Peterson, “Quantification of Thermal Contact Conductance in Electronic Packages,” in *Fifth Annual IEEE Semiconductor Thermal and Temperature Measurement Symposium (SEMI-THERM)*, San Diego, CA, USA, pp. 30–36, February 1989.
- [42] D. J. Dean, *Thermal Design of Electronic Circuit Boards and Packages*, Ayr, Scotland: Electrochemical Publications Limited, 1985.
- [43] G. N. Ellison, “Theoretical Calculation of the Thermal Resistance of a Conducting and Convecting Surface,” *IEEE Transactions on Parts, Hybrids, and Packaging*, vol. PHP-12, no. 3, pp. 265–266, 1976.
- [44] S. N. Rea and S. E. West, “Thermal Radiation from Finned Heat Sinks,” *IEEE Transactions on Parts, Hybrids, and Packaging*, vol. PHP-12, no. 2, pp. 115–117, 1976.
- [45] L. Buller and B. McNelis, “Effects of Radiation on Enhanced Electronic Cooling,” *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, vol. 11, no. 4, pp. 538–544, 1988.
- [46] C. Zardini *et al.*, “3D Thermal Simulation of Power Hybrid Assemblies,” *Hybrid Circuits*, no. 24, pp. 20–22, January 1991.
- [47] I. Hirsch, E. Berman, and N. Haik, “Thermal Resistance Evaluation in 3-D Thermal Simulation of MOSFET Transistors,” *Solid-State Electronics*, vol. 36, no. 1, pp. 106–108, 1993.

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 (refer to the *Sentaurus Device Monte Carlo User Guide*). In Sentaurus Device, the band structure is simplified to four quantities: the energies of the conduction and valence band edges (or, in a different parameterization, band gap and electron affinity), and the density-of-states masses for electrons and holes (or, parameterized differently, the band edge density-of-states). The models for these quantities are discussed in [Band Gap and Electron Affinity](#) and [Effective Masses and Effective Density-of-States](#) on page 237.

Intrinsic Density

The band gap and band edge density-of-states are summarized in the intrinsic density $n_i(T)$ (for undoped semiconductors):

$$n_i(T) = \sqrt{N_C(T)N_V(T)} \exp\left(-\frac{E_g(T)}{2kT}\right) \quad (128)$$

and the effective intrinsic density (including doping-dependent bandgap narrowing):

$$n_{i,\text{eff}} = n_i \exp\left(\frac{E_{\text{bgn}}}{2kT}\right) \quad (129)$$

In devices that contain different materials, the electron affinity χ (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`, `delAlamo`, `OldSlotboom`, and `Slotboom` (the same model with different default parameters), `JainRoulston`, and `TableBGN`. The bandgap model can be selected in the `EffectiveIntrinsicDensity` statement in the `Physics` section, for example:

```
Physics {
    EffectiveIntrinsicDensity(BandGapNarrowing (Slotboom))
}
```

activates the `Slotboom` model. The default model is `BennettWilson`.

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 26 on page 236](#) and [Table 27 on page 236](#) list the model parameters available for calibration (see [Bandgap and Electron-Affinity Models](#)).

Bandgap and Electron-Affinity Models

Sentaurus Device models the lattice temperature-dependence of the band gap as [1]:

$$E_g(T) = E_g(0) - \frac{\alpha T^2}{T + \beta} \quad (130)$$

where $E_g(0)$ is the bandgap energy at 0 K, and α and β are material parameters (see [Table 26 on page 236](#)).

To allow $E_g(0)$ to differ for different bandgap models, it is written as:

$$E_g(0) = E_{g,0} + \delta E_{g,0} \quad (131)$$

$E_{g,0}$ 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 26 on page 236](#)).

The effective band gap results from the band gap reduced by bandgap narrowing:

$$E_{g,\text{eff}}(T) = E_g(T) - E_{\text{bgn}} \quad (132)$$

The electron affinity χ 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:

$$\chi(T) = \chi_0 + \frac{\alpha T^2}{2(T+\beta)} + \text{Bgn2Chi} \cdot E_{\text{bgn}} \quad (133)$$

where χ_0 and Bgn2Chi are adjustable parameters (see [Table 26 on page 236](#)). Bgn2Chi defaults to 0.5 and, therefore, bandgap narrowing splits equally between conduction and valence bands.

In the case of the JainRoulston model, the electron affinity depends also on the doping concentration (especially at high dopings) because bandgap narrowing is doping dependent:

$$\chi(T, N_{A,0}, N_{D,0}) = \chi_0 + \frac{\alpha T^2}{2(T+\beta)} + E_{\text{bgn}}^{\text{cond}} \quad (134)$$

where $E_{\text{bgn}}^{\text{cond}}$ is the shift in the conduction band due to the JainRoulston bandgap narrowing and χ_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:

$$E_{\text{bgn}} = \Delta E_g^0 + \Delta E_g^{\text{Fermi}} \quad (135)$$

where ΔE_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 [Eq. 139](#)).

Bandgap Narrowing for Bennett–Wilson Model

Bandgap narrowing for the Bennett–Wilson [2] model (keyword `BennettWilson`) in Sentaurus Device reads:

$$\Delta E_g^0 = \begin{cases} E_{\text{ref}} \left[\ln \left(\frac{N_{\text{tot}}}{N_{\text{ref}}} \right) \right]^2 & N_{\text{tot}} \geq N_{\text{ref}} \\ 0 & \text{otherwise} \end{cases} \quad (136)$$

5: 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_{ref} and N_{ref} are accessible in the `Bennett` parameter set in the parameter file of Sentaurus Device (see [Table 27 on page 236](#)).

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:

$$\Delta E_g^0 = E_{\text{ref}} \left[\ln\left(\frac{N_{\text{tot}}}{N_{\text{ref}}}\right) + \sqrt{\left(\ln\left(\frac{N_{\text{tot}}}{N_{\text{ref}}}\right)\right)^2 + 0.5} \right] \quad (137)$$

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_{ref} and N_{ref} are accessible in the `Slotboom` and `oldSlotboom` parameter sets in the parameter file (see [Table 27 on page 236](#)).

Bandgap Narrowing for del Alamo Model

Bandgap narrowing for the del Alamo model (keyword `delAlamo`) in Sentaurus Device reads:

$$\Delta E_g^0 = \begin{cases} E_{\text{ref}} \ln\left(\frac{N_{\text{tot}}}{N_{\text{ref}}}\right) & N_{\text{tot}} \geq N_{\text{ref}} \\ 0 & \text{otherwise} \end{cases} \quad (138)$$

This model was proposed [7]–[11] for n-type materials. The material parameters E_{ref} and N_{ref} are accessible in the `delAlamo` parameter set in the parameter file (see [Table 27 on page 236](#)).

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:

$$\Delta E_g^0 = A \cdot N_{\text{tot}}^{1/3} + B \cdot N_{\text{tot}}^{1/4} + C \cdot N_{\text{tot}}^{1/2} + D \cdot N_{\text{tot}}^{1/2} \quad (139)$$

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]:

$$A = 1.83 \frac{\Lambda}{N_b^{1/3}} \frac{aR}{\left(\frac{3}{4\pi}\right)^{1/3}} \quad [\text{eV} \cdot \text{cm}] \quad (140)$$

$$B = \left((0.95Ra^{3/4}) / \left(\frac{3}{4\pi}\right)^{1/4} \right) \quad [\text{eV} \cdot \text{cm}^{3/4}] \quad (141)$$

$$C = \frac{1.57Ra^{3/2}}{N_b \left(\frac{3}{4\pi}\right)^{1/2}} \quad [\text{eV} \cdot \text{cm}^{3/2}] \quad (142)$$

$$D = \frac{1.57R_{(\text{mino})}a^{3/2}}{N_b \left(\frac{3}{4\pi}\right)^{1/2}} \quad [\text{eV} \cdot \text{cm}^{3/2}] \quad (143)$$

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 Λ 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 Eq. 139, 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 $C_3 = B + D$.

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 B and D using Eq. 142 and Eq. 143. Sentaurus Device needs four coefficients instead of three to compute $E_{\text{bgn}}^{\text{cond}}$ and the doping-dependent affinity (Eq. 134) internally. The expression of $E_{\text{bgn}}^{\text{cond}}$ is given by:

$$E_{\text{bgn}}^{\text{cond}} = \begin{cases} A \cdot N_{\text{tot}}^{1/3} + C \cdot N_{\text{tot}}^{1/2} & N_{\text{D},0} > N_{\text{A},0} \\ B \cdot N_{\text{tot}}^{1/4} + D \cdot N_{\text{tot}}^{1/2} & \text{otherwise} \end{cases} \quad (144)$$

5: Semiconductor Band Structure

Band Gap and Electron Affinity

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]
        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)]
    }
}
```

The default values of the coefficients are listed in [Table 24](#).

Table 24 Default parameters for Jain–Roulston bandgap narrowing model

Symbol	Parameter name	Default value for material				Unit
		Si	Ge	GaAs	All others	
A	A_n	1.02e-8	7.30e-9	1.65e-8	0.0	eVcm
	A_p	1.11e-8	8.21e-9	9.77e-9	0.0	eVcm
B	B_n	4.15e-7	2.57e-7	2.38e-7	0.0	$eVcm^{3/4}$
	B_p	4.79e-7	2.91e-7	3.87e-7	0.0	$eVcm^{3/4}$
C	C_n	1.45e-12	2.29e-12	1.83e-11	0.0	$eVcm^{3/2}$
	C_p	3.23e-12	3.58e-12	3.41e-12	0.0	$eVcm^{3/2}$
D	D_n	1.48e-12	2.03e-12	7.25e-11	0.0	$eVcm^{3/2}$
	D_p	1.81e-12	2.19e-12	4.84e-13	0.0	$eVcm^{3/2}$

Table Specification of Bandgap Narrowing

It is possible to specify bandgap narrowing by 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.

Each table entry is a line that specifies a concentration type (Donor, Acceptor, or Total) with a concentration in 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} cm^{-3} , the bandgap narrowing vanishes, then increases up to 20 meV at 10^{20} 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} 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 It is not possible to 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, OldSlotboom, Slotboom, JainRoulston, 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.

5: Semiconductor Band Structure

Band Gap and Electron Affinity

The Schenk bandgap narrowing model described in [13] 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 `ISimplified=-1` in the Schenk bandgap narrowing section of the parameter file.

The full-model exchange correlation and ionic terms are described by:

$$\Delta_a^{\text{xc}}(n, p, T) = -\frac{(4\pi)^3 n_{\Sigma}^2 \left[\left(\frac{48n_a}{\pi g_a} \right)^{1/3} + c_a \ln(1 + d_a n_p^{p_a}) \right] + \left(\frac{8\pi\alpha_a}{g_a} \right) n_a \Upsilon^2 + \sqrt{8\pi n_{\Sigma}} \Upsilon^{5/2}}{(4\pi)^3 n_{\Sigma}^2 + \Upsilon^3 + b_a \sqrt{n_{\Sigma}} \Upsilon^2 + 40n_{\Sigma}^{3/2} \Upsilon} \quad (145)$$

$$\Delta_a^{\text{i}}(N_{\text{tot}}, n, p, T) = -\frac{N_{\text{tot}} [1 + U^{\text{i}}(n_{\Sigma}, \Upsilon)]}{\sqrt{\Upsilon n_{\Sigma}/(2\pi)} [1 + h_a \ln(1 + \sqrt{n_{\Sigma}}/\Upsilon) + j_a U^{\text{i}}(n_{\Sigma}, \Upsilon) n_p^{3/4} (1 + k_a n_p^{q_a})]} \quad (146)$$

and, for the simplified model by:

$$\Delta_a^{\text{xc}}(n, p, T=0) = -\left[\left(\frac{48n_a}{\pi g_a} \right)^{1/3} + c_a \ln(1 + d_a n_p^{p_a}) \right] \quad (147)$$

$$\Delta_a^{\text{i}}(N_{\text{tot}}, n, p, T=0) = -N_{\text{tot}} \frac{0.799\alpha_a}{n_p^{3/4}} \quad (148)$$

where $a = e, h$ is the index for the carrier type, $\Upsilon = kT/Ry_{\text{ex}}$, $n_{\Sigma} = n + p$, $n_p = \alpha_e n + \alpha_h p$, and $U^{\text{i}}(n_{\Sigma}, \Upsilon) = n_{\Sigma}^2/\Upsilon^3$. The default values (silicon) and units for the parameters that can be changed in Sentaurus Device in this model are listed in [Table 25 on page 234](#).

Sentaurus Device implements the Schenk bandgap narrowing model using the framework of the density gradient model (see [Density Gradient Model on page 260](#)). If you want to suppress the quantization corrections contained in this model, set the density gradient model parameter $\gamma = 0$ (see [Eq. 185, p. 260](#)).

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 Λ_{PMI} in [Eq. 185, p. 260](#)) for `eQuantumPotential` (conduction bandgap correction), and `SchenkBGN_hole` must be specified as the `LocalModel` for `hQuantumPotential` (valence band correction). 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, γ 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
    }
}
```

5: Semiconductor Band Structure

Band Gap and Electron Affinity

```

        }
SchenkBGN_hole {
    ...
    * Selects simplified model (1) or complete Schenk model (-1)
    IsSimplified = 1
}
}

```

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 [13] and their default values are summarized in [Table 25](#).

The Schenk bandgap narrowing can be visualized by specifying the data entry `eSchenkBGN` or `hSchenkBGN` or both in the `Plot` section of the input file.

Table 25 Default parameters for Schenk bandgap narrowing model

Symbol	Parameter name	Default value (silicon)	Unit
SchenkBGN_elec			
α_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
SchenkBGN_hole			
α_h	alpha_h	0.4813	1
g_h	g_h	4	1
Ry_{ex}	Ry_ex	16.55	meV

Table 25 Default parameters for Schenk bandgap narrowing model

Symbol	Parameter name	Default value (silicon)	Unit
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 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.

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 [Eq. 135](#)):

$$\Delta E_g^{\text{Fermi}} = k300K \left[\ln\left(\frac{N_V N_C}{N_{A,0} N_{D,0}}\right) + F_{1/2}^{-1}\left(\frac{N_{A,0}}{N_V}\right) + F_{1/2}^{-1}\left(\frac{N_{D,0}}{N_C}\right) \right] \quad (149)$$

where the right-hand side is evaluated at 300 K.

By default, correction [Eq. 149](#) 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

5: Semiconductor Band Structure

Band Gap and Electron Affinity

assuming Fermi statistics. Sometimes for III–IV materials, the correction Eq. 149 is too large, and it is recommended to switch it off in these cases.

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 χ_0 and the temperature coefficients α and β . As an extension to what Eq. 130 and Eq. 133 suggest, the parameters $E_{g,0}$ and χ_0 can be specified at any reference temperature T_{par} . By default, $T_{\text{par}} = 0\text{ K}$.

Table 26 Bandgap models: Default parameters 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	–	Eq. 131
$\delta E_{g,0}$	dEg0 (Bennett)	0.0	eV	1.1696	
	dEg0 (Slotboom)	-4.795×10^{-3}	eV	1.1648	
	dEg0 (OldSlotboom)	-1.595×10^{-2}	eV	1.1537	
	dEg0 (delAlamo)	-1.407×10^{-2}	eV	1.1556	
α	alpha	4.73×10^{-4}	eV /K	–	Eq. 130, Eq. 133
β	beta	636	K	–	Eq. 130, Eq. 133
χ_0	Chi0	4.05	eV	–	Eq. 133
Bgn2Chi	Bgn2Chi	0.5	l	–	Eq. 133
T_{par}	Tpar	0	K	–	

Table 27 summarizes the silicon default parameters for the analytic bandgap narrowing models available in Sentaurus Device.

Table 27 Bandgap narrowing models: Default parameters for silicon

Symbol	Parameter	Bennett	Slotboom	Old Slotboom	del Alamo	Unit
E_{ref}	Ebgn	6.84×10^{-3}	6.92×10^{-3}	9.0×10^{-3}	18.7×10^{-3}	eV
N_{ref}	Nref	3.162×10^{18}	1.3×10^{17}	1.0×10^{17}	7.0×10^{17}	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 density-of-states (DOS) as a function of carrier effective mass. The effective mass may 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 density-of-states. The default for simulations of GaAs devices is `Formula=2`.

Electron Effective Mass and DOS

Formula 1

The lattice temperature-dependence of the DOS effective mass of electrons is modeled by:

$$m_n = 6^{2/3} (m_t^2 m_l)^{1/3} + m_m \quad (150)$$

where the temperature-dependent effective mass component $m_t(T)$ is best described in silicon by the temperature dependence of the energy gap [14]:

$$\frac{m_t(T)}{m_0} = a \frac{E_g(0)}{E_g(T)} \quad (151)$$

The coefficient a and the mass m_l are defined in the parameter file with the default values provided in [Table 28 on page 238](#). The parameter m_m , which defaults to zero, allows m_n to be defined as a temperature-independent quantity if required.

The effective densities of states (DOS) in the conduction band N_C follows from:

$$N_C(m_n, T_n) = 2.5094 \times 10^{19} \left(\frac{m_n}{m_0} \right)^{\frac{3}{2}} \left(\frac{T_n}{300 \text{ K}} \right)^{\frac{3}{2}} \text{cm}^{-3} \quad (152)$$

5: Semiconductor Band Structure

Effective Masses and Effective Density-of-States

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:

$$N_C(T_n) = N_C(300\text{ K}) \left(\frac{T_n}{300\text{ K}} \right)^{\frac{3}{2}} \quad (153)$$

and the electron effective mass is simply a function of $N_C(300\text{ K})$:

$$\frac{m_n}{m_0} = \left(\frac{N_C(300\text{ K})}{2.5094 \times 10^{19} \text{ cm}^{-3}} \right)^{\frac{2}{3}} \quad (154)$$

Electron Effective Mass and Conduction Band DOS Parameters

Table 28 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 28 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>	2.890×10^{19}	cm^{-3}

Hole Effective Mass and DOS

Formula 1

For the DOS effective mass of holes, the best fit in silicon is provided by the expression [15]:

$$\frac{m_p(T)}{m_0} = \left(\frac{a + bT + cT^2 + dT^3 + eT^4}{1 + fT + gT^2 + hT^3 + iT^4} \right)^{\frac{2}{3}} + m_m \quad (155)$$

where the coefficients are listed in [Table 29 on page 240](#). 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:

$$N_V(m_p, T_p) = 2.5094 \times 10^{19} \left(\frac{m_p}{m_0} \right)^{\frac{3}{2}} \left(\frac{T_p}{300\text{K}} \right)^{\frac{3}{2}} \text{cm}^{-3} \quad (156)$$

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:

$$N_V(T_p) = N_V(300\text{K}) \left(\frac{T_p}{300\text{K}} \right)^{\frac{3}{2}} \quad (157)$$

and the effective hole mass is given by:

$$\frac{m_p}{m_0} = \left(\frac{N_V(300\text{K})}{2.5094 \times 10^{19} \text{cm}^{-3}} \right)^{\frac{2}{3}} \quad (158)$$

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 29](#) 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 29 Default coefficients for hole effective mass and DOS models

Option	Symbol	Parameter name	Electrons	Unit
Formula=1	a	a	0.4435870	1
	b	b	0.3609528×10^{-2}	K^{-1}
	c	c	0.1173515×10^{-3}	K^{-2}
	d	d	0.1263218×10^{-5}	K^{-3}
	e	e	0.3025581×10^{-8}	K^{-4}
	f	f	0.4683382×10^{-2}	K^{-1}
	g	g	0.2286895×10^{-3}	K^{-2}
	h	h	0.7469271×10^{-6}	K^{-3}
	i	i	0.1727481×10^{-8}	K^{-4}
m _m	mm		0	1
Formula=2	N _V (300 K)	Nv300	3.140×10^{19}	cm^{-3}

Gaussian Density-of-States for Organic Semiconductors

Gaussian density-of-states (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:

$$\Gamma(E) = \frac{N_t}{\sqrt{2\pi}\sigma_{\text{DOS}}} \exp\left(-\frac{(E-E_0)^2}{2\sigma_{\text{DOS}}^2}\right) \quad (159)$$

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), and E_0 and σ_{DOS} are the energy center and width of the DOS distribution, respectively.

Introducing the notations:

$$\begin{aligned}
 A_e &= \frac{\sigma_{\text{DOS}, e}^2}{2k^2} \\
 B_e &= \frac{E_0 - E_C}{k} \\
 C_e &= \frac{E_0 - E_C}{\sqrt{2}\sigma_{\text{DOS}, e}} \\
 A_h &= \frac{\sigma_{\text{DOS}, h}^2}{2k^2} \\
 B_h &= \frac{E_V - E_0}{k} \\
 C_h &= \frac{E_V - E_0}{\sqrt{2}\sigma_{\text{DOS}, h}}
 \end{aligned} \tag{160}$$

the electron and hole effective DOS are computed in the Maxwell–Boltzmann approximation as:

$$N_C(T) = \frac{N_{\text{LUMO}}}{2} \exp\left(\frac{A_e}{T^2} - \frac{B_e}{T}\right) \operatorname{erfc}\left(\frac{\sqrt{A_e}}{T} - C_e\right) \tag{161}$$

and:

$$N_V(T) = \frac{N_{\text{HOMO}}}{2} \exp\left(\frac{A_h}{T^2} - \frac{B_h}{T}\right) \operatorname{erfc}\left(\frac{\sqrt{A_h}}{T} - C_h\right) \tag{162}$$

The 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_sem1") {
    EffectiveMass(GaussianDOS)
}
```

5: Semiconductor Band Structure

References

The model parameters used in [Eq. 159](#) and [Eq. 160](#) and their default values are summarized in [Table 30](#).

Table 30 Gaussian DOS model parameters

Parameter symbol	Parameter name	Default value	Unit
N_t	N_{LUMO} (electron)	1×10^{21}	cm^{-3}
	N_{HOMO} (hole)	1×10^{21}	
σ_{DOS}	$\sigma_{\text{DOS}, e}$	0.2	eV
	$\sigma_{\text{DOS}, h}$	0.2	
$E_{0c} = E_0 - E_C$	E0_c	0.1	eV
$E_{0v} = E_V - E_0$	E0_v	0.1	

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 $\text{Ge}_x\text{Si}_{1-x}$ Strained Layers," *Solid-State Electronics*, vol. 34, no. 5, pp. 453–465, 1991.
- [13] 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.
- [14] 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.
- [15] 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.

5: Semiconductor Band Structure

References

This chapter discusses how incomplete ionization is accounted for in Sentaurs Device.

Overview

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, Sentaurs Device has an ionization probability model based on activation energy. The ionization (activation) is computed separately for each species present.

Sentaurs Device supports the most significant dopants used in silicon technologies: the donors As, P, Sb, and N, and the acceptors B and In. For the simulation of other semiconductors such as III–V compounds, the actual donors and acceptors used (Si, Be, and so on) are not supported. However, it is reasonable to replace the real dopant species with an appropriate silicon donor or acceptor and adjust the physical parameters accordingly.

For example, to simulate GaAs, which is doped n-type using Si, P can be substituted as the donor. The donor level and cross-sections of P should then be changed in the parameter file to represent the values for silicon in GaAs. Alternatively, Sentaurs Device supports the generic dopants ‘NDopant’ and ‘PDopant.’

The doping profiles for these generic dopants are read from the doping file under the names NDopantConcentration and PDopantConcentration.

Using Incomplete Ionization

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") }
```

The incomplete ionization model can be specified in region or material physics (see [Region-specific and Material-specific Physics on page 55](#)). In this case, the model is activated only in these regions or materials.

Incomplete Ionization Model

The concentration of ionized impurity atoms is given by Fermi–Dirac distribution:

$$N_D = \frac{N_{D,0}}{1 + g_D \exp\left(\frac{E_{F,n} - E_D}{kT}\right)} \text{ for } N_{D,0} < N_{D,\text{crit}} \quad (163)$$

$$N_A = \frac{N_{A,0}}{1 + g_A \exp\left(\frac{E_A - E_{F,p}}{kT}\right)} \text{ for } N_{A,0} < N_{A,\text{crit}} \quad (164)$$

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:

$$N_D = \frac{N_{D,0}}{1 + G_D(T) \exp\left(\frac{E_{F,n} - E_C}{kT}\right)} \quad (165)$$

$$N_A = \frac{N_{A,0}}{1 + G_A(T) \exp\left(-\frac{E_{F,p} - E_V}{kT}\right)} \quad (166)$$

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 Model on page 1017](#)).

By comparing [Eq. 163](#) and [Eq. 164](#) to [Eq. 165](#) and [Eq. 166](#), it can be seen that, in the case of the Fermi distribution function, the ionization factors can be written as:

$$G_D(T) = g_D \cdot \exp\left(\frac{\Delta E_D}{kT}\right), \Delta E_D = E_C - E_D \text{ and } G_A(T) = g_A \cdot \exp\left(\frac{\Delta E_A}{kT}\right), \Delta E_A = E_A - E_V \quad (167)$$

In Sentaurus Device, the basic variables are potential, electron concentration, and hole concentration. Therefore, it is more convenient to rewrite [Eq. 163](#) and [Eq. 164](#) in terms of the carrier concentration instead of the quasi-Fermi levels:

$$N_D = \frac{N_{D,0}}{1 + g_D \frac{n}{n_1}}, \text{ with } n_1 = N_C \exp\left(-\frac{\Delta E_D}{kT}\right) \text{ for } N_{D,0} < N_{D,\text{crit}} \quad (168)$$

$$N_A = \frac{N_{A,0}}{1 + g_A \frac{p}{p_1}}, \text{ with } p_1 = N_V \exp\left(-\frac{\Delta E_A}{kT}\right) \text{ for } N_A < N_{A,\text{crit}} \quad (169)$$

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 7 on page 251](#)) is used, n_1 and p_1 are multiplied by the coefficients γ_n and γ_p defined in [Eq. 90](#) and [Eq. 91](#) (see [Fermi Statistics on page 202](#)). 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.

The donor and acceptor activation energies are effectively reduced by the total doping in the semiconductor. This effect is accounted for in the expressions:

$$\Delta E_D = \Delta E_{D,0} - \alpha_D \cdot N_{\text{tot}}^{1/3} \quad (170)$$

$$\Delta E_A = \Delta E_{A,0} - \alpha_A \cdot N_{\text{tot}}^{1/3} \quad (171)$$

where $N_{\text{tot}} = N_{A,0} + N_{D,0}$ is the total doping concentration.

In transient simulations, the terms:

$$\frac{\partial N_D}{\partial t} = \sigma_D v_{\text{th}}^n \left[\frac{n_1}{g_D} N_{D,0} - \left(n + \frac{n_1}{g_D} \right) N_D \right] \quad (172)$$

$$\frac{\partial N_A}{\partial t} = \sigma_A v_{\text{th}}^p \left[\frac{p_1}{g_A} N_{A,0} - \left(p + \frac{p_1}{g_A} \right) N_A \right] \quad (173)$$

are included in the continuity equations ($v_{\text{th}}^{n,p}$ denote the carrier thermal velocities).

Physical Model Parameters

The values of the dopant level $E_{A/D,0}$, the doping-dependent shift parameter $\alpha_{A/D}$, the impurity degeneracy factor $g_{A/D}$, and the cross section $\sigma_{A/D}$ are accessible in the parameter set `Ionization`.

For each user-defined dopant (see [User-Defined Species on page 125](#)), the `Ionization` parameter set must contain a separate subsection where the parameters $E_{A/D,0}$, $\alpha_{A/D}$, $g_{A/D}$, and $\sigma_{A/D}$ are defined. For example, for the dopant Nitrogen described in [User-Defined Species on page 125](#) for the material SiC, the parameter set is:

```
Material = "SiC" {
    ...
    Ionization {
        ...
        Species("Nitrogen") {
            type = donor
            E_0 = 0.1
            alpha = 0
            g = 2
            Xsec = 1.0e-15
        }
    }
}
```

The field `type` can be omitted because the dopant type is specified in the `datexcodes.txt` file. It is used here for informative purposes only.

Table 31 Default coefficients for incomplete ionization model for dopants in silicon

Symbol	Parameter name	Default value for species *								Unit
		As	P	Sb	B	In	N	NDopant	PDopant	
$E_{A/D, 0}$	E_*_0	0.054	0.045	0.039	0.045	0.16	0.045	0.045	0.045	eV
$\alpha_{A/D}$	alpha_*	3.1×10^{-8}								eVcm
$g_{A/D}$	g_*	2	2	2	4	4	2	2	4	1
$\sigma_{A/D}$	$Xsec_*$	1.0×10^{-12}								$\text{cm}^2 \text{s}^{-1}$
$N_{D,crit}$	$NdCrit$	1.0×10^{22}								cm^{-3}
$N_{A,crit}$	$NaCrit$	1.0×10^{22}								cm^{-3}

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.

6: Incomplete Ionization

References

Quantization Models

This chapter describes the features that Sentaurus Device offers to model quantization effects.

Some features of current MOSFETs (oxide thickness, 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 17 on page 511](#).

Overview

To include quantization effects in a classical device simulation, Sentaurus Device introduces a potential-like quantity Λ_n in the classical density formula:

$$n = N_C F_{1/2} \left(\frac{E_{F,n} - E_C - \Lambda_n}{kT_n} \right) \quad (174)$$

An analogous quantity Λ_p is introduced for holes.

The most important effects related to the density modification (due to quantization) can be captured by proper models for Λ_n and Λ_p . Other effects (for example, single electron effects) exceed the scope of this approach.

Sentaurus Device implements four quantization models, that is, four different models for Λ_n and Λ_p . They differ in physical sophistication, numeric expense, and robustness:

- The van Dort model (see [van Dort Quantization Model on page 252](#)) is a numerically robust, fast, and proven model. It is only suited 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 (see [1D Schrödinger Solver on page 253](#)) is the most physically sophisticated quantization model. 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 current flow. Therefore, the Schrödinger equation is used mainly for the validation and calibration of other quantization models.

7: Quantization Models

van Dort Quantization Model

- The density gradient model (see [Density Gradient Quantization Model on page 260](#)) 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 on page 264](#)) 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 three-dimensional device simulations because of its numeric efficiency.

van Dort Quantization Model

van Dort Model

The van Dort model [1] computes Λ_n of [Eq. 174](#) as a function of $|\hat{n} \cdot \vec{F}|$, the electric field normal to the semiconductor–insulator interface:

$$\Lambda_n = \frac{13}{9} \cdot k_{\text{fit}} \cdot G(\vec{r}) \cdot \left(\frac{\epsilon \epsilon_0}{4kT} \right)^{1/3} \cdot |\hat{n} \cdot \vec{F} - E_{\text{crit}}|^{2/3} \quad (175)$$

and likewise for Λ_p . k_{fit} and E_{crit} are fitting parameters.

The function $G(\vec{r})$ is defined by:

$$G(\vec{r}) = \frac{2 \cdot \exp(-a^2(\vec{r}))}{1 + \exp(-2a^2(\vec{r}))} \quad (176)$$

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 λ_{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 the `eQCvanDort` or `hQCvanDort` flag in the `Physics` section:

```
Physics { ... eQCvanDort }
```

[Table 32](#) lists the parameters in the `vanDortQMModel` parameter set. The default value of λ_{ref} is from the literature [1]. Other parameters were obtained by fitting the model to experimental data for electrons [1] and holes [2].

Table 32 Default parameters for van Dort quantum correction model

Symbol	Electrons		Holes		Unit
k_{fit}	<code>eFit</code>	2.4×10^{-8}	<code>hFit</code>	1.8×10^{-8}	eVcm
E_{crit}	<code>eEcritQC</code>	10^5	<code>eEcritQC</code>	10^5	V/cm
λ_{ref}	<code>dRef</code>	2.5×10^{-6}	<code>dRef</code>	2.5×10^{-6}	cm

By default, in [Eq. 175](#), \hat{n} is the normal to the closest semiconductor–insulator interface. The interface can be specified explicitly by `EnormalInterface` in the `Math` section (see [Device-specific Math Keywords on page 92](#)).

1D Schrödinger Solver

The 1D Schrödinger solver implements the most physically sophisticated quantization model in Sentaurus Device. To use the Schrödinger solver:

1. Construct a special purpose ‘nonlocal’ mesh (see [Nonlocal Mesh for 1D Schrödinger on page 254](#)).
2. Activate the Schrödinger solver on the nonlocal line mesh with appropriate parameters (see [Using 1D Schrödinger on page 255](#)).

Sometimes, especially for heteromaterials, the physical model parameters need to be adapted (see [1D Schrödinger Parameters on page 255](#)).

NOTE The 1D Schrödinger solver is time consuming and often causes converge problems. Furthermore, small-signal analysis (see [ACCoupled: Small-Signal AC Analysis on page 161](#)) and noise and fluctuation analysis (see [Chapter 16 on page 499](#)) are not possible when using this solver.

Nonlocal Mesh for 1D Schrödinger

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 104](#).

To generate a nonlocal mesh, specify `NonLocal` in the global `Math` section. Options to `NonLocal` control the construction of the nonlocal mesh. For example:

```
Math "NLM" {
    NonLocal(
        RegionInterface="gateoxide/channel"
        Length=10e-7
        Permeation=1e-7
        Direction=(0 1 0) MaxAngle=5
        -Transparent(Region="gateoxide")
    )
}
```

generates a nonlocal mesh named "NLM" at the interface between region `gateoxide` and `channel`. 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.

For nonlocal meshes constructed for the Schrödinger equation, the `-Transparent` switch is important. In the above 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 1D Schrödinger

To activate the 1D Schrödinger equation for electrons or holes, specify the `Electron` or `Hole` switch as an option to `Schroedinger` in the global `Physics` section. For example:

```
Physics {  
    Schroedinger "NLM" (Electron)  
}
```

activates the Schrödinger equation for electrons on the nonlocal mesh named "`NLM`".

Additional options to `Schroedinger` determine numeric accuracy, details of the density computation, and which eigenstates will be computed. [Table 221 on page 1175](#) 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.

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 110](#)).

1D Schrödinger Parameters

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, 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.

Explicit Ladder Specification

Any number of `eLadder`($m_{z,v}$, $m_{xy,v}$, d_v , ΔE_v) (for electrons) or `hLadder`($m_{z,v}$, $m_{xy,v}$, d_v , ΔE_v) (for holes) specifications is possible in the `SchroedingerParameters` parameter set. 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 ΔE_v .

The parameter pair `ShiftTemperature` in the `SchroedingerParameters` parameter set is given in kelvin and determines the temperatures T_{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:

$$\frac{m_n^{3/2}}{\sum_v d_v m_{xy,v} m_{z,v}^{1/2} \exp(-\Delta E_v / k T_{\text{ref}})} \quad (177)$$

and likewise for holes. T_{ref} defaults to a large value. When $T_{\text{ref}} = 0$, scaling is disabled.

Automatic Extraction of Ladder Parameters

When no explicit ladder specification is present, Sentaurus Device attempts to extract the required parameters automatically. The `formula` parameter pair in the `SchroedingerParameters` parameter set specifies which expressions for the masses to use.

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. The quantization mass for the conduction band valley v reads:

$$\frac{1}{m_{z,v}} = \sum_{i=1}^3 \frac{z_i^2}{m_{i,v}} \quad (178)$$

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 Reference System on page 576](#)).

For holes, ‘warped’ band structure (formula 1), or heavy-hole and light-hole masses (formula 2) are available (see [Table 33](#)). For the former, the quantization mass is given by:

$$m_{z,v} = \frac{m_0}{A \pm \sqrt{B + C \cdot (z_1^2 z_2^2 + z_2^2 z_3^2 + z_3^2 z_1^2)}} \quad (179)$$

For the latter, m_l and m_h (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:

$$m_{xy,v} = \frac{m_n^{3/2}}{m_{z,v}^{1/2} n_v} \quad (180)$$

where n_v is the number of band valleys (or bands). The expression for holes is analogous.

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 [Eq. 177](#).

Table 33 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	m _l	0	1
	m_h	m _h	0	1

Visualizing Schrödinger Solutions

To visualize the results obtained by the Schrödinger equation, Sentaurus Device offers special keywords for the NonLocalPlot section (see [Visualizing Data Defined on Nonlocal Meshes on page 106](#)).

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 [Eq. 181](#)).

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. For example:

```
NonLocal(
    (0 0)
) {
    WaveFunction(Electron(Number=3) Hole)
}
```

will plot all hole wavefunctions and the three lowest-energy electron wavefunctions for the nonlocal mesh line close to the coordinate (0, 0, 0).

1D Schrödinger Model

Omitting x- and y-coordinates for simplicity, the 1D Schrödinger equation for electrons is:

$$\left(-\frac{\partial^2}{\partial z^2} \frac{\hbar^2}{2m_{z,v}(z)} + E_C(z) + \Delta E_v \right) \Psi_{j,v}(z) = E_{j,v} \Psi_{j,v}(z) \quad (181)$$

where z is the quantization direction (typically, the direction perpendicular to the silicon–oxide interface in a MOSFET), v labels the ladder, ΔE_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, and $E_{j,v}$ is the j -th eigenenergy.

From the solution of this equation, the density is computed as:

$$n(z) = \frac{kT(z)}{\pi\hbar^2} \sum_{j,v} |\Psi_{j,v}(z)|^2 d_v m_{xy,v}(z) \left[F_0\left(\frac{E_{F,n}(z) - E_{j,v}}{kT(z)}\right) - F_0\left(\frac{E_{F,n}(z) - E_{\max,v}}{kT(z)}\right) \right] + \sum_v n_{cl,v} \quad (182)$$

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 ΔE_v , see [1D Schrödinger Parameters on page 255](#).

If `DensityTail=MaxEnergy`, $E_{\max,v}$ in Eq. 182 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 Eq. 182. Equating the resulting density to Eq. 174 determines Λ_n .

The Schrödinger equation is solved over a finite domain $[z_-, z_+]$. At the endpoints of this domain, the boundary condition:

$$\frac{\Psi'_{j,v}}{\Psi_{j,v}} = \mp \frac{\sqrt{2m_{z,v}|E_{j,v} - E_C|}}{\hbar} \quad (183)$$

is applied, where for the upper sign, all position-dependent functions are taken at z_+ and, for the lower sign, at z_- .

1D Schrödinger Application Notes

- 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.
- 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 1D Schrödinger on page 254](#)). 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.

7: Quantization Models

Density Gradient Quantization Model

Density Gradient Quantization Model

Density Gradient Model

For the density gradient model [3][4], Λ_n in Eq. 174 is given by a partial differential equation:

$$\Lambda_n = -\frac{\gamma \hbar^2}{12m_n} \left\{ \nabla^2 \ln n + \frac{1}{2} (\nabla \ln n)^2 \right\} = -\frac{\gamma \hbar^2}{6m_n} \frac{\nabla^2 \sqrt{n}}{\sqrt{n}} \quad (184)$$

where γ is a fit factor.

Introducing the reciprocal thermal energy $\beta = 1/kT_n$, the mass-driving term $\Phi_m = -kT_n \ln(N_C/N_{ref})$ (with an arbitrary normalization constant N_{ref}) and the smoothed potential $\Phi = E_C + \Phi_m + \Lambda_n$, Eq. 184 can be rewritten and generalized as:

$$\begin{aligned} \Lambda_n = & -\frac{\hbar^2 \gamma}{12m_n} \{ \nabla \cdot \alpha(\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi) + \\ & 9(\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi) \cdot \alpha(\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi)^2 \} + \Lambda_{PMI} \end{aligned} \quad (185)$$

with the default parameters $\xi = \eta = 1$, $\vartheta = 1/2$, and α 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 Eq. 185, which for Fermi statistics deviates from Eq. 184, even when $\xi = \eta = 1$ and $\vartheta = 1/2$. The density-based expression Eq. 184 is available as an option (see [Using the Density Gradient Model on page 261](#)).

In Eq. 185, Λ_{PMI} is a locally dependent quantity computed from a user-specified PMI model (see [Apparent Band-Edge Shift on page 970](#)), from the Schenk bandgap narrowing model (see [Schenk Bandgap Narrowing Model on page 231](#)), or from apparent band-edge shifts caused by multistate configurations (see [Apparent Band-Edge Shift on page 368](#)). By default, $\Lambda_{PMI} = 0$.

At Ohmic contacts, interfaces to metals with Ohmic boundary conditions, resistive contacts, and current contacts, the boundary condition $\Lambda_n = \Lambda_{PMI}$ 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, $\bar{\Phi}$ and $\hat{n} \cdot \alpha(\xi \nabla \beta E_{F,n} - \nabla \beta \bar{\Phi} + (\eta - 1)q \nabla \beta \phi)$ 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:

$$\hat{n} \cdot \nabla \Lambda_n = \sqrt{\frac{24m_n(0^+)k^3T_n^3}{\hbar^2\gamma(0^+)\vartheta(0^+)^2\alpha}} [E_C(0^+) - E_C(0^-) - \Lambda_n] f\left(\frac{2\vartheta(0^+)}{kT_n} [\Lambda_n - E_C(0^+) + E_C(0^-)]\right) \quad (186)$$

where $f(x) = \sqrt{[(\exp x - 1)/x - 1]/x}$ and $\bar{\alpha} = \det[\alpha(0^+)]^{1/3}$.

Optionally, Sentaurus Device offers a modified mobility to improve the modeling of tunneling through semiconductor barriers:

$$\mu = \frac{\mu_{cl} + r\mu_{tunnel}}{1 + r} \quad (187)$$

where μ_{cl} is the usual (classical) mobility as described in [Chapter 8 on page 267](#), μ_{tunnel} is a fit parameter, and $r = \max(0, n/n_{cl} - 1)$. Here, n_{cl} is the ‘classical’ density (see [Eq. 298](#)). In this modification, for $n > n_{cl}$, the additional carriers (density $n - n_{cl}$) are considered as tunneling carriers that are subject to a different mobility μ_{tunnel} than the classical carriers (density n_{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 `-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 208 on page 1166](#).

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 [Eq. 184](#) instead of the potential-based formula [Eq. 185](#). If this option is used, the parameters ξ and η must both be 1.

The `LocalModel` option of `eQuantumPotential` and `hQuantumPotential` specifies the name of a PMI model (see [Physical Model Interface on page 911](#)) or the Schenk bandgap

7: Quantization Models

Density Gradient Quantization Model

narrowing model (see [Schenk Bandgap Narrowing Model on page 231](#)) for Λ_{PMI} in [Eq. 185](#). 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 368](#)).

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
}
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 [Eq. 187](#), specify the Tunneling switch to Mobility in the Physics section of the command file. Specify μ_{tunnel} for electrons and holes by the mutunnel parameter pair in the ConstantMobility parameter set.

The parameters γ , ϑ , ξ , and η 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, ξ is always assumed as zero, regardless of user specifications.

The diagonal of α in the coordinate system specified by the LatticeParameters parameter set (see [Crystal Reference System on page 576](#)) is determined by the parameter pairs alpha[1], alpha[2], and alpha[3] (for the x -, y -, and z -direction, respectively) in the QuantumPotentialParameters parameter set. Unless α is a multiple of the unit matrix, Sentaurus Device solves an anisotropic problem. The anisotropic density gradient equation supports the AverageAniso approximation only (see [Chapter 21 on page 575](#)).

Density Gradient Application Notes

- Fitting parameters: The parameter γ 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) may require recalibration.
- 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 17 on page 511](#)). To suppress unwanted tunneling or to fit tunneling currents despite these concerns, consider using the modified mobility model according to [Eq. 187](#).
- 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 [Coupled Command on page 65](#)) 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 202](#)). 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.

- 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 601](#)). 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 Δm_n is the change in mass due to strain. The parameter v can be specified in the QuantumPotentialParameters parameter set.

Modified Local-Density Approximation

MLDA Model

The modified local-density approximation (MLDA) model is a quantum-mechanical model that calculates the confined carrier distributions that occur near Si–SiO₂ 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 run-time.

Following Paasch and Übensee [5], the confined electron density at a distance z from a Si–SiO₂ interface is given, under Fermi statistics, by:

$$n_{\text{MLDA}}(\eta_n) = N_C \left(\frac{2}{\sqrt{\pi}} \right) \int_0^{\infty} d\eta \frac{\sqrt{\eta}}{1 + \exp[(\eta - \eta_n)]} [1 - j_0(2z\sqrt{\eta}/\lambda_n)] \quad (188)$$

where η_n is given by Eq. 92, j_0 is the 0th-order spherical Bessel function, and $\lambda_n = \sqrt{\hbar^2/2m_n kT_n}$ is the electron thermal wavelength. The integrand of Eq. 188 is very similar to the classical Fermi integrand, with an additional factor describing confinement for small z . The electron temperature is used to calculate η_n . Accordingly, Λ_n of Eq. 174 is calculated from the confined electron density. Holes are treated similarly.

Under Boltzmann statistics, Eq. 188 simplifies to the following expression for the electron density (the hole density is similar):

$$n_{\text{MLDA}}(\eta_n) = N_C \exp(\eta_n) [1 - \exp(-(z/\lambda_n)^2)] \quad (189)$$

Using MLDA

To activate the model, use the `MLDA` switch in the `Physics` section. To activate the model for electrons or holes only, `eMLDA` or `hMLDA` can be used. The model can also be specified in the regionwise or materialwise physics section, and can be switched off by using `-MLDA`, `-eMLDA`, or `-hMLDA`.

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 (
        * eMLDA | hMLDA for one carrier
        -LambdaTemp      * turn off temperature dependence
    )
}
```

Table 34 lists the coefficients for this model. The default values of λ_{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 34 Default coefficients for MLDA model

Symbol	Electrons	Holes	Unit
λ_{300}	eLambda	23.5×10^{-8}	cm

By default, the MLDA model looks for all semiconductor-insulator interfaces and calculates the normal distance z in [Eq. 188](#) and [Eq. 189](#) from the nearest interface. The MLDA model also uses the EnormalInterface option (see [Discretization Methods on page 93](#)) and the GeometricDistances option (see [Normal to Interface on page 286](#)). In addition, 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.

MLDA Application Notes

- Due to the nature of the MLDA model, the spurious roll-off of the carrier density can occur at the semiconductor–insulator interface inside the drain/source diffusion. Then, MLDAbox can be used to confine the application of the model only to the inversion layer under the gate of a MOS structure.
- The MLDA model is a robust and efficient quantum-mechanical model, which can be considered as an alternative to the more time-consuming but accurate density gradient model. However, it must be noted that the MLDA model is based on the one-dimensional and the first-order approximation with the assumption of the triangular well with infinite barrier. In particular, it fails to calculate the more classical carrier density near the drain when the device is operating in the saturation mode.
- 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.

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. Hareland 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.

This chapter presents information about the different mobility models implemented in Sentaurus Device.

Sentaurus Device uses a modular approach for the description of the carrier mobilities. In the simplest case, the mobility is a function of the lattice temperature. This so-called constant mobility model described in [Mobility due to Phonon Scattering on page 268](#) should only be used for undoped materials. For doped materials, the carriers scatter with the impurities. This leads to a degradation of the mobility. [Doping-dependent Mobility Degradation on page 268](#) introduces the models that describe this effect.

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 273](#). These models account for the scattering with surface phonons and surface roughness.

Models that describe the effects of carrier–carrier scattering are given in [Carrier–Carrier Scattering on page 288](#). The Philips unified mobility model described in [Philips Unified Mobility Model on page 289](#) is a well-calibrated model, which accounts for both impurity and carrier–carrier scattering.

Finally, the models that describe mobility degradation in high electric fields are discussed in [High-Field Saturation on page 294](#). A flexible model for hydrodynamic simulations is described in [Energy-dependent Mobility on page 568](#).

How Mobility Models Combine

The 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.

8: Mobility Models

Mobility due to Phonon Scattering

If more than one mobility model is activated for a carrier type, the different mobility contributions are combined according to the following scheme – different bulk (μ_{b1} , μ_{b2} , ...) and surface (μ_{s1} , μ_{s2} , ...) mobility contributions are combined following Matthiessen's rule:

$$\frac{1}{\mu} = \frac{1}{\mu_{b1}} + \frac{1}{\mu_{b2}} + \dots + \frac{1}{\mu_{s1}} + \frac{1}{\mu_{s2}} + \dots \quad (190)$$

If the high-field saturation model is activated, the final mobility is computed in two steps. First, the low field mobility μ_{low} is determined according to Eq. 190. Second, the final mobility is computed from a (model-dependent) formula as a function of a driving force F_{hfs} :

$$\mu = f(\mu_{low}, F_{hfs}) \quad (191)$$

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:

$$\mu_{const} = \mu_L \left(\frac{T}{300K} \right)^{-\zeta} \quad (192)$$

where μ_L is the mobility due to bulk phonon scattering. The default values of μ_L and the exponent ζ are listed in Table 35. The constant mobility model parameters are accessible in the ConstantMobility parameter set.

Table 35 Constant mobility model: Default coefficients for silicon

Symbol	Parameter name	Electrons	Holes	Unit
μ_L	mumax	1417	470.5	cm ² /Vs
ζ	exponent	2.5	2.2	1

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 three built-in models and two PMIs for doping-dependent mobility.

Using Doping-dependent Mobility

The models for the mobility degradation due to impurity scattering are activated by specifying the `DopingDependence` flag to `Mobility`:

```
Physics{ Mobility( DopingDependence ... ) ... }
```

Different models are available and are selected by options to `DopingDependence`:

```
Physics{ Mobility( DopingDependence( [ Masetti | Arora | UniBo ] ) ... ) ... }
```

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 the variable `formula`, which is accessible in the `DopingDependence` parameter set:

```
DopingDependence: {
    formula= 1 , 1 # [1]
    ...
}
```

If `formula` is set to 1, the Masetti model is the default. To use the Arora model as the default, set `formula` to 2.

The `DopingDependence` parameter set also determines the other parameters for the Masetti and Arora models. The parameters for the University of Bologna model are in the `UniBoDopingDependence` parameter set.

Masetti Model

The default model used by Sentaurus Device to simulate doping-dependent mobility in silicon was proposed by Masetti *et al.* [2]:

$$\mu_{\text{dop}} = \mu_{\text{min1}} \exp\left(-\frac{P_c}{N_{A,0} + N_{D,0}}\right) + \frac{\mu_{\text{const}} - \mu_{\text{min2}}}{1 + ((N_{A,0} + N_{D,0})/C_r)^\alpha} - \frac{\mu_1}{1 + (C_s/(N_{A,0} + N_{D,0}))^\beta} \quad (193)$$

The reference mobilities μ_{min1} , μ_{min2} , and μ_1 , the reference doping concentrations P_c , C_r , and C_s , and the exponents α and β are accessible in the parameter set `DopingDependence`.

8: Mobility Models

Doping-dependent Mobility Degradation

The corresponding values for silicon are given in [Table 36](#).

Table 36 Masetti model: Default coefficients

Symbol	Parameter name	Electrons	Holes	Unit
$\mu_{\min 1}$	mumin1	52.2	44.9	cm^2/Vs
$\mu_{\min 2}$	mumin2	52.2	0	cm^2/Vs
μ_1	mu1	43.4	29.0	cm^2/Vs
P_c	Pc	0	9.23×10^{16}	cm^{-3}
C_r	Cr	9.68×10^{16}	2.23×10^{17}	cm^{-3}
C_s	Cs	3.34×10^{20}	6.10×10^{20}	cm^{-3}
α	alpha	0.680	0.719	1
β	beta	2.0	2.0	1

The low-doping reference mobility μ_{const} is determined by the constant mobility model (see [Mobility due to Phonon Scattering on page 268](#)).

Arora Model

The Arora model [3] reads:

$$\mu_{\text{dop}} = \mu_{\min} + \frac{\mu_d}{1 + ((N_{A,0} + N_{D,0})/N_0)^{A^*}} \quad (194)$$

with:

$$\mu_{\min} = A_{\min} \cdot \left(\frac{T}{300 \text{ K}} \right)^{\alpha_m}, \quad \mu_d = A_d \cdot \left(\frac{T}{300 \text{ K}} \right)^{\alpha_d} \quad (195)$$

and:

$$N_0 = A_N \cdot \left(\frac{T}{300 \text{ K}} \right)^{\alpha_N}, \quad A^* = A_a \cdot \left(\frac{T}{300 \text{ K}} \right)^{\alpha_a} \quad (196)$$

The parameters are accessible in the DopingDependence parameter set.

Table 37 Arora model: Default coefficients for silicon

Symbol	Parameter name	Electrons	Holes	Unit
A_{\min}	Ar_mumin	88	54.3	cm ² /Vs
α_m	Ar_alm	-0.57	-0.57	1
A_d	Ar_mud	1252	407	cm ² /Vs
α_d	Ar_ald	-2.33	-2.23	1
A_N	Ar_N0	1.25×10^{17}	2.35×10^{17}	cm ⁻³
α_N	Ar_alN	2.4	2.4	1
A_a	Ar_a	0.88	0.88	1
α_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 284](#)). 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:

$$\mu_L(T) = \mu_{\max} \left(\frac{T}{300 \text{ K}} \right)^{-\gamma + c \left(\frac{T}{300 \text{ K}} \right)} \quad (197)$$

where μ_{\max} denotes the lattice mobility at room temperature, and c gives a correction to the lattice mobility at higher temperatures. The maximum mobility μ_{\max} and the exponents γ and c are accessible in the parameter file.

8: Mobility Models

Doping-dependent Mobility Degradation

The model for bulk mobility reads:

$$\mu_{\text{dop}}(T) = \mu_0(T) + \frac{\mu_L(T) - \mu_0(T)}{1 + \left(\frac{N_{D,0}}{C_{r1}(T)}\right)^{\alpha} + \left(\frac{N_{A,0}}{C_{r2}(T)}\right)^{\beta}} - \frac{\mu_1(N_{D,0}, N_{A,0}, T)}{1 + \left(\frac{N_{D,0}}{C_{s1}(T)} + \frac{N_{A,0}}{C_{s2}(T)}\right)^{-2}} \quad (198)$$

In turn, μ_0 and μ_1 are expressed as weighted averages of the corresponding limiting values for pure acceptor-doping and pure donor-doping densities:

$$\mu_0(T) = \frac{\mu_{0d}N_{D,0} + \mu_{0a}N_{A,0}}{N_{A,0} + N_{D,0}} \quad (199)$$

$$\mu_1(T) = \frac{\mu_{1d}N_{D,0} + \mu_{1a}N_{A,0}}{N_{A,0} + N_{D,0}} \quad (200)$$

The reference mobilities μ_{0d} , μ_{0a} , μ_{1d} , and μ_{1a} , and the reference doping concentrations C_{r1} , C_{r2} , C_{s1} , and C_{s2} are accessible in the parameter file.

Table 38 lists the corresponding values of silicon for arsenic, phosphorus, and boron; $T_n = T/300$ K.

Table 38 Parameters of University of Bologna bulk mobility model

Silicon	Parameter name	Electrons (As)	Electrons (P)	Holes (B)	Unit
μ_{max}	mumax	1441	1441	470.5	cm^2/Vs
c	Exponent2	-0.11	-0.11	0	1
γ	Exponent	2.45	2.45	2.16	1
μ_{0d}	mumin1	$55.0T_n^{-\gamma_{0d}}$	$62.2T_n^{-\gamma_{0d}}$	$90.0T_n^{-\gamma_{0d}}$	cm^2/Vs
γ_{0d}	mumin1_exp	0.6	0.7	1.3	1
μ_{0a}	mumin2	$132.0T_n^{-\gamma_{0a}}$	$132.0T_n^{-\gamma_{0a}}$	$44.0T_n^{-\gamma_{0a}}$	cm^2/Vs
γ_{0a}	mumin2_exp	1.3	1.3	0.7	1
μ_{1d}	mu1	$42.4T_n^{-\gamma_{1d}}$	$48.6T_n^{-\gamma_{1d}}$	$28.2T_n^{-\gamma_{1d}}$	cm^2/Vs
γ_{1d}	mu1_exp	0.5	0.7	2.0	1
μ_{1a}	mu2	$73.5T_n^{-\gamma_{1a}}$	$73.5T_n^{-\gamma_{1a}}$	$28.2T_n^{-\gamma_{1a}}$	cm^2/Vs
γ_{1a}	mu2_exp	1.25	1.25	0.8	1
C_{r1}	Cr	$8.9 \times 10^{16}T_n^{\gamma_{r1}}$	$8.5 \times 10^{16}T_n^{\gamma_{r1}}$	$1.3 \times 10^{18}T_n^{\gamma_{r1}}$	cm^{-3}
γ_{r1}	Cr_exp	3.65	3.65	2.2	1
C_{r2}	Cr2	$1.22 \times 10^{17}T_n^{\gamma_{r2}}$	$1.22 \times 10^{17}T_n^{\gamma_{r2}}$	$2.45 \times 10^{17}T_n^{\gamma_{r2}}$	cm^{-3}

Table 38 Parameters of University of Bologna bulk mobility model

Silicon	Parameter name	Electrons (As)	Electrons (P)	Holes (B)	Unit
γ_{r2}	Cr2_exp	2.65	2.65	3.1	1
C_{s1}	Cs	$2.9 \times 10^{20} T_n^{\gamma_{s1}}$	$4.0 \times 10^{20} T_n^{\gamma_{s1}}$	$1.1 \times 10^{18} T_n^{\gamma_{s1}}$	cm^{-3}
γ_{s1}	Cs_exp	0	0	6.2	1
C_{s2}	Cs2	7.0×10^{20}	7.0×10^{20}	6.1×10^{20}	cm^{-3}
α	alpha	0.68	0.68	0.77	1
β	beta	0.72	0.72	0.719	1

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.

PMIs for Bulk Mobility

The PMIs for bulk mobility are described in [Doping-dependent Mobility on page 942](#) and [Multistate Configuration-dependent Bulk Mobility on page 947](#).

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 286](#)).

To select the calculation of field perpendicular to the semiconductor–insulator interface, specify the Enormal option to Mobility:

```
Physics{ Mobility( Enormal ... ) ... }
```

8: Mobility Models

Mobility Degradation at Interfaces

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, Lombardi_highk, IALMob, UniBo, or a PMI model provided by you. For example,

```
Physics{ Mobility( Enormal(UniBo) ... ) ... }
```

selects the University of Bologna model (see [University of Bologna Surface Mobility Model on page 284](#)). The default model is Lombardi (see [Enhanced Lombardi Model on page 274](#)).

NOTE The mobility degradation models discussed in this section are very sensitive to mesh spacing. It is recommended that the vertical mesh spacing be reduced to 0.1 nm in the silicon at the oxide interface underneath the gate. For the extensions of the Lombardi model (see [Eq. 204](#)), even smaller spacing of 0.05 nm is appropriate. This fine spacing is only required in the two uppermost rows of mesh and can be increased progressively moving away from the interface.

Enhanced Lombardi Model

The surface contribution due to acoustic phonon scattering has the form:

$$\mu_{ac} = \frac{B}{F_{\perp}} + \frac{C((N_{A,0} + N_{D,0})/N_0)^{\lambda}}{F_{\perp}^{1/3}(T/300\text{K})^k} \quad (201)$$

and the contribution attributed to surface roughness scattering is given by:

$$\mu_{sr} = \left(\frac{(F_{\perp}/F_{ref})^{A^*}}{\delta} + \frac{F_{\perp}^3}{\eta} \right)^{-1} \quad (202)$$

These surface contributions to the mobility (μ_{ac} and μ_{sr}) are then combined with the bulk mobility μ_b according to Matthiessen's rule (see [Mobility due to Phonon Scattering on page 268](#) and [Doping-dependent Mobility Degradation on page 268](#)):

$$\frac{1}{\mu} = \frac{1}{\mu_b} + \frac{D}{\mu_{ac}} + \frac{D}{\mu_{sr}} \quad (203)$$

The reference field $F_{ref} = 1 \text{ V/cm}$ ensures a unitless numerator in [Eq. 202](#). F_{\perp} is the transverse electric field normal to the semiconductor-insulator interface, see [Computing Transverse Field on page 286](#). $D = \exp(-x/l_{crit})$ (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. All other parameters are accessible in the parameter file.

In the Lombardi model [1], the exponent A^* in Eq. 202 is equal to 2. According to another study [6], an improved fit to measured data is achieved if A^* is given by:

$$A^* = A + \frac{\alpha_{\perp}(n+p)N_{\text{ref}}^v}{(N_{A,0} + N_{D,0} + N_1)^v} \quad (204)$$

where n and p denote the electron and hole concentrations, respectively. 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 Eq. 204. The Lombardi model parameters are accessible in the parameter set `EnormalDependence`.

The respective default parameters that are appropriate for silicon are given in Table 39. The parameters B , C , N_0 , and λ were fitted at SGS Thomson and are not contained in the literature [1].

Table 39 Lombardi model: Default coefficients for silicon

Symbol	Parameter name	Electrons	Holes	Unit
B	B	4.75×10^7	9.925×10^6	cm/s
C	C	5.80×10^2	2.947×10^3	$\text{cm}^{5/3}\text{V}^{-2/3}\text{s}^{-1}$
N_0	N0	1	1	cm^{-3}
λ	lambda	0.1250	0.0317	1
k	k	1	1	1
δ	delta	5.82×10^{14}	2.0546×10^{14}	cm^2/Vs
A	A	2	2	1
α_{\perp}	alpha	0	0	cm^3
N_1	n1	1	1	cm^{-3}
v	nu	1	1	1
η	eta	5.82×10^{30}	2.0546×10^{30}	$\text{V}^2\text{cm}^{-1}\text{s}^{-1}$
l_{crit}	l_crit	1×10^{-6}	1×10^{-6}	cm

The modifications of the Lombardi model suggested in [6] can be activated by setting α_{\perp} to a nonzero value.

8: Mobility Models

Mobility Degradation at Interfaces

Table 40 lists a consistent set of parameters for the modified model.

Table 40 Lombardi model: Lucent coefficients for silicon

Symbol	Parameter name	Electrons	Holes	Unit
B	B	3.61×10^7	1.51×10^7	cm/s
C	C	1.70×10^4	4.18×10^3	$\text{cm}^{5/3}\text{V}^{-2/3}\text{s}^{-1}$
N_0	n0	1	1	cm^{-3}
λ	lambda	0.0233	0.0119	1
k	k	1.7	0.9	1
δ	delta	3.58×10^{18}	4.10×10^{15}	cm^2/Vs
A	A	2.58	2.18	1
α_{\perp}	alpha	6.85×10^{-21}	7.82×10^{-21}	cm^3
N_1	n1	1	1	cm^{-3}
v	nu	0.0767	0.123	1
η	eta	1×10^{50}	1×10^{50}	$\text{V}^2\text{cm}^{-1}\text{s}^{-1}$
l_{crit}	l_crit	1	1	cm

Enhanced Lombardi Model with High-k Degradation

High-k gate dielectrics are being considered as an alternative to SiO_2 to reduce unacceptable leakage currents as transistor dimensions become smaller. 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, two possible contributors are remote Coulomb scattering (RCS) and remote phonon scattering (RPS).

Sentaurus Device supports a version of the Lombardi model (see [Enhanced Lombardi Model on page 274](#)) that includes empirical degradation terms accounting for RCS and RPS. These are combined with the Lombardi model using Matthiessen's rule:

$$\frac{1}{\mu} = \frac{1}{\mu_{\text{Lombardi}}} + \alpha_{\text{rcs}} \cdot \frac{D_{\text{rcs}} D_{\text{rcs_highk}}}{\Delta \mu_{\text{rcs}}} + \alpha_{\text{rps}} \cdot \frac{D_{\text{rps}} D_{\text{rps_highk}}}{\Delta \mu_{\text{rps}}} \quad (205)$$

The $\Delta \mu_{\text{rcs}}$ term is taken from [7] and accounts for the mobility degradation observed with HfSiON MISFETs. This term is mostly attributed to RCS.

$$\Delta\mu_{\text{rcs}} = \mu_{\text{rcs}} \left(\frac{N_{\text{A,D}}}{3 \times 10^{16} \text{ cm}^{-3}} \right)^{\gamma_1} \left(\frac{T}{300 \text{ K}} \right)^{\gamma_2} \left(\frac{N_s}{10^{11} \text{ cm}^{-2}} \right)^{\gamma_3 + \gamma_4 \cdot \ln\left(\frac{N_{\text{A,D}}}{3 \times 10^{16} \text{ cm}^{-3}}\right)} \times f(F_{\perp}) \quad (206)$$

In Eq. 206, N_s is the inversion carrier density [cm^{-2}] and $f(F_{\perp})$ is a function that confines the degradation to areas of the structure where F_{\perp} is not small:

$$f(F_{\perp}) = 1 - \exp(-1.3 \times 10^{13} \text{ V}^{-1} \text{ cm}^{-1} \cdot F_{\perp} / N_{\text{depl}}) \quad (207)$$

The $\Delta\mu_{\text{rps}}$ term is extracted from figures in [8] and accounts for a portion of the mobility degradation observed with HfO_2 -gated MOSFETs. This term is mostly attributed to RPS:

$$\Delta\mu_{\text{rps}} = \mu_{\text{rps}} \left(\frac{F_{\perp}}{10^6 \text{ V/cm}} \right)^{\gamma_5} \left(\frac{T}{300 \text{ K}} \right)^{\gamma_6} \quad (208)$$

The α_{rcs} and α_{rps} parameters in Eq. 205 can be used to emphasize or de-emphasize the degradation components. A value of 1 for these parameters includes the full degradation term. A value of zero disables the degradation term.

The degradation terms include two different types of damping factor. The D_{rcs} and D_{rps} factors in Eq. 205 are damping factors that reduce the degradation components based on the distance measured from the semiconductor–insulator interface. The $D_{\text{rcs_highk}}$ and $D_{\text{rps_highk}}$ factors reduce the degradation components based on the distance measured from any high-k insulator found in the structure:

$$D_{\text{rcs}} = \exp(-(dist + d_{\text{crit,rcs}}) / l_{\text{crit,rcs}}) \quad (209)$$

$$D_{\text{rps}} = \exp(-(dist + d_{\text{crit,rps}}) / l_{\text{crit,rps}}) \quad (210)$$

$$D_{\text{rcs_highk}} = \exp(-dist_{\text{highk}} / l_{\text{crit,rcs_highk}}) \quad (211)$$

$$D_{\text{rps_highk}} = \exp(-dist_{\text{highk}} / l_{\text{crit,rps_highk}}) \quad (212)$$

In these expressions, $dist$ is the distance from the semiconductor–insulator interface and $dist_{\text{highk}}$ is the distance from any high-k insulator found in the structure. If no high-k insulator is found in the structure, the damping factors $D_{\text{rcs_highk}}$ and $D_{\text{rps_highk}}$ are ignored.

The enhanced Lombardi model with high-k degradation can be selected by specifying the parameter `Lombardi_highk` in the command file, for example:

```
Physics{ Mobility( Enormal(Lombardi_highk) ... ) ... }
```

8: Mobility Models

Mobility Degradation at Interfaces

Model parameters associated with the model are accessible in the Lombardi_highk parameter set. Their values for silicon are shown in [Table 41 to Table 43 on page 279](#).

Table 41 Lombardi parameters for Lombardi_highk model: Default coefficients for silicon

Symbol	Electron parameter name	Electron value	Hole parameter name	Hole value	Unit
B	B_e	4.75×10^7	B_h	9.925×10^6	cm/s
C	C_e	5.80×10^2	C_h	2.947×10^3	$\text{cm}^{5/3}\text{V}^{-2/3}\text{s}^{-1}$
λ	lambda_e	0.1250	lambda_h	0.0317	1
k	k_e	1	k_h	1	1
δ	delta_e	5.82×10^{14}	delta_h	2.0546×10^{14}	cm ² /Vs
A	A_e	2	A_h	2	1
α_{\perp}	alpha_e	0	alpha_h	0	cm ³
v	nu_e	1	nu_h	1	1
η	eta_e	5.82×10^{30}	eta_h	2.0546×10^{30}	V ² cm ⁻¹ s ⁻¹
l_{crit}	l_crit_e	1×10^{-6}	l_crit_h	1×10^{-6}	cm

Table 42 RCS parameters for Lombardi_highk model: Default coefficients for silicon

Symbol	Electron parameter name	Electron value	Hole parameter name	Hole value	Unit
α_{rcs}	alpha_rcs_e	1.0	alpha_rcs_h	1.0	1
μ_{rcs}	mu_rcs_e	240.0	mu_rcs_h	240.0	cm ² /Vs
γ_1	N1_exp_e	-0.30	N1_exp_h	-0.30	1
γ_2	T1_exp_e	2.1	T1_exp_h	2.1	1
γ_3	Ns1_exp_e	0.40	Ns1_exp_h	0.40	1
γ_4	Ns2_exp_e	0.035	Ns2_exp_h	0.035	1
$d_{\text{crit,rcs}}$	d_crit_rcs_e	0.0	d_crit_rcs_h	0.0	cm
$l_{\text{crit,rcs}}$	l_crit_rcs_e	1×10^{-6}	l_crit_rcs_h	1×10^{-6}	cm
$l_{\text{crit,rcs_highk}}$	l_crit_rcs_highk_e	1×10^6	l_crit_rcs_highk_h	1×10^6	cm

Table 43 RPS parameters for Lombardi_highk model: Default coefficients for silicon

Symbol	Electron parameter name	Electron value	Hole parameter name	Hole value	Unit
α_{rps}	alpha_rps_e	0.0	alpha_rps_h	0.0	1
μ_{rps}	mu_rps_e	496.7	mu_rps_h	496.7	cm ² /Vs
γ_5	En_exp_e	-0.68	En_exp_h	-0.68	1
γ_6	T2_exp_e	-0.34	T2_exp_h	-0.34	1
$d_{\text{crit,rps}}$	d_crit_rps_e	0.0	d_crit_rps_h	0.0	cm
$l_{\text{crit,rps}}$	l_crit_rps_e	1×10^{-6}	l_crit_rps_h	1×10^{-6}	cm
$l_{\text{crit,rps_highk}}$	l_crit_rps_highk_e	1×10^6	l_crit_rps_highk_h	1×10^6	cm

NOTE The Lombardi_highk model has its own set of Lombardi parameters (see [Table 41 on page 278](#)). The default values of these parameters are the same as the standard Lombardi model (see [Table 39 on page 275](#)). In addition, the default hole parameter values for the RCS and RPS components of the model are the same as the default electron parameter values. Some calibration of these parameters is necessary to obtain accurate results for a specific technology.

Inversion and Accumulation Layer Mobility Model

The inversion and accumulation layer mobility model [9] is similar to the Lucent (Darwish) model [6] but contains additional terms that account for ‘two-dimensional’ Coulomb impurity scattering in the inversion and accumulation regions of a MOSFET. Although this model is based on modified versions of the [Philips Unified Mobility Model on page 289](#) and the [Enhanced Lombardi Model on page 274](#), it is completely self-contained and all parameters associated with this model are specified independently of those models. A complete description is given here.

In the expressions that follow, the donor and acceptor concentrations are obtained from the clustering formulas used in the Philips unified mobility model:

$$N_D^* = N_{D,0} \left[1 + \frac{N_{D,0}^2}{c_D N_{D,0}^2 + N_{D,\text{ref}}^2} \right] \quad (213)$$

$$N_A^* = N_{A,0} \left[1 + \frac{N_{A,0}^2}{c_A N_{A,0}^2 + N_{A,\text{ref}}^2} \right] \quad (214)$$

8: Mobility Models

Mobility Degradation at Interfaces

$$N_{\text{total}} = N_{\text{D}}^* + N_{\text{A}}^* \quad (215)$$

The expressions that follow also apply to both electron and hole mobility when the following substitutions are made:

$$\begin{aligned} \text{Electron Mobility: } c &= n, N_{\text{inv}} = N_{\text{A}}^*, N_{\text{acc}} = N_{\text{D}}^*, P = P_e \\ \text{Hole Mobility: } c &= p, N_{\text{inv}} = N_{\text{D}}^*, N_{\text{acc}} = N_{\text{A}}^*, P = P_h \end{aligned} \quad (216)$$

The doping-dependent and transverse field portion of the inversion and accumulation layer mobility model has contributions from acoustic phonon scattering, surface roughness scattering, and Coulomb impurity scattering:

$$\frac{1}{\mu} = \frac{1}{\mu_{\text{ph}}} + \frac{1}{\mu_{\text{sr}}} + \frac{1}{\mu_C} \quad (217)$$

The phonon scattering portion of Eq. 209 has 2D and 3D parts:

$$\mu_{\text{ph}} = \min\{\mu_{\text{ph},2D}, \mu_{\text{ph},3D}\} \quad (218)$$

where:

$$\mu_{\text{ph},2D} = \frac{B}{F_{\perp}} + \frac{C(N_{\text{total}}/1 \text{ cm}^{-3})^{\lambda}}{F_{\perp}^{1/3}(T/300 \text{ K})^k} \quad (219)$$

$$\mu_{\text{ph},3D} = \mu_{\text{max}} \left(\frac{T}{300 \text{ K}} \right)^{-\theta} \quad (220)$$

The surface roughness scattering is given by:

$$\mu_{\text{sr}} = \frac{(N_{\text{total}}/1 \text{ cm}^{-3})^{\lambda}}{\left(\frac{(F_{\perp}/1 \text{ V/cm})^{A^*}}{\delta} + \frac{F_{\perp}^3}{\eta} \right)} \quad (221)$$

where:

$$A^* = A + \frac{\alpha_{\perp}(n+p)}{(1+N_{\text{total}}/1 \text{ cm}^{-3})^v} \quad (222)$$

The Coulomb impurity scattering term has 2D and 3D parts that are combined using a normal field-dependent Fermi function:

$$\mu_C = f(\alpha_C)\mu_{C,3D} + (1 - f(\alpha_C))\mu_{C,2D} \quad (223)$$

where:

$$f(\alpha_C) = \frac{1}{1 + \exp(2\alpha_C - 4)} \quad (224)$$

$$\alpha_C = (0.1521 \text{ cm}^{2/3} \text{ K/V}^{2/3}) F_\perp^{2/3} / T \quad (225)$$

The two-dimensional Coulomb scattering part contains both the accumulation and inversion layer contributions:

$$\frac{1}{\mu_{C,2D}} = \frac{1}{\mu_{C,2D,acc}} + \frac{1}{\mu_{C,2D,inv}} \quad (226)$$

where:

$$\mu_{C,2D,inv} = \max \left\{ \frac{D_1 \cdot (c/10^{18} \text{ cm}^{-3})^{v_0}}{(N_{inv}/10^{18} \text{ cm}^{-3})^{v_1}}, \frac{D_2}{(N_{inv}/10^{18} \text{ cm}^{-3})^{v_2}} \right\} \quad (227)$$

$$\mu_{C,2D,acc} = \max \left\{ \frac{D_1 \cdot (c/10^{18} \text{ cm}^{-3})^{v_0}}{(N_{acc}/10^{18} \text{ cm}^{-3})^{v_1}}, \frac{D_2}{(N_{acc}/10^{18} \text{ cm}^{-3})^{v_2}} \right\} G(P) \quad (228)$$

The 3D Coulomb scattering part of Eq. 215 is given by:

$$\mu_{C,3D} = \mu_N \left(\frac{N_{total}}{N_{sc,eff}} \right) \left(\frac{N_{ref}}{N_{total}} \right)^\alpha + \mu_c \left(\frac{c}{N_{sc,eff}} \right) \quad (229)$$

where:

$$\mu_N = \frac{\mu_{max}^2}{\mu_{max} - \mu_{min}} \left(\frac{T}{300K} \right)^{3\alpha - 1.5} \quad (230)$$

$$\mu_c = \frac{\mu_{max} \mu_{min}}{\mu_{max} - \mu_{min}} \left(\frac{300K}{T} \right)^{1/2} \quad (231)$$

$$N_{sc,eff} = N_{acc} + G(P)N_{inv} \quad (232)$$

8: Mobility Models

Mobility Degradation at Interfaces

The function $G(P)$ is the Philips unified mobility model scattering ratio, but here, the $G = G_{\min}$ restriction for $P < P_{\min}$ is ignored, and P is the Philips unified mobility model screening term, but here, the ‘other’ carrier is ignored:

$$G(P) = 1 - \frac{0.89233}{\left[0.41372 + P\left(\frac{m_0}{m^*} \frac{T}{300K}\right)^{0.28227}\right]^{0.19778}} + \frac{0.005978}{\left[P\left(\frac{m^*}{m_0} \frac{300K}{T}\right)^{0.72169}\right]^{1.80618}} \quad (233)$$

$$P = \left[\frac{2.459}{3.97 \times 10^{13} \text{ cm}^{-2} N_{\text{total}}^{2/3}} + \frac{3.828 c(m_0/m^*)}{1.36 \times 10^{20} \text{ cm}^{-3}} \right]^{-1} \left(\frac{T}{300K} \right)^2 \quad (234)$$

Parameters associated with the model are accessible in the IALMob parameter set. Their values for silicon are shown in [Table 44](#) to [Table 46 on page 283](#).

Table 44 IALMob parameters for clustering: Default coefficients for silicon

Symbol	Donor parameter name	Donor value	Acceptor parameter name	Acceptor value	Unit
$N_{\{D,A\},\text{ref}}$	nref_D	4×10^{20}	nref_A	7.2×10^{20}	cm^{-3}
$c_{\{D,A\}}$	cref_D	0.21	cref_A	0.5	1

Table 45 IALMob parameters for phonon and surface roughness scattering: Default coefficients for silicon

Symbol	Electron parameter name	Electron value	Hole parameter name	Hole value	Unit
B	B_e	9.0×10^5	B_h	9.0×10^5	cm/s
C	C_e	4400.0	C_h	4400.0	$\text{cm}^{5/3} \text{V}^{-2/3} \text{s}^{-1}$
λ	lambda_e	0.057	lambda_h	0.057	1
k	k_e	1	k_h	1	1
δ	delta_e	3.97×10^{13}	delta_h	3.97×10^{13}	cm^2/Vs
η	eta_e	1.0×10^{50}	eta_h	1.0×10^{50}	$\text{V}^2 \text{cm}^{-1} \text{s}^{-1}$
A	A_e	2	A_h	2	1
α_{\perp}	alpha_sr_e	0	alpha_sr_h	0	cm^3
v	nu_e	0	nu_h	0	1

Table 46 IALMob parameters for lattice and Coulomb scattering: Default coefficients for silicon

Symbol	Electron parameter name	Electron value	Hole parameter name	Hole value	Unit
μ_{\max}	mumax_e	1417.0	mumax_h	470.5	cm ² /Vs
μ_{\min}	mumin_e	52.2	mumin_h	44.9	cm ² /Vs
θ	theta_e	2.285	theta_h	2.247	1
N_{ref}	n_ref_e	9.68×10^{16}	n_ref_h	2.23×10^{17}	cm ⁻³
α	alpha_e	0.68	alpha_h	0.719	1
m^*/m_0	me_over_m0	0.26	mh_over_m0	0.26	1
D_1	D1_e	135.0	D1_h	135.0	cm ² /Vs
D_2	D2_e	40.0	D2_h	40.0	cm ² /Vs
v_0	nu0_e	1.5	nu0_h	1.5	1
v_1	nu1_e	2.0	nu1_h	2.0	1
v_2	nu2_e	0.5	nu2_h	0.5	1

Using Inversion and Accumulation Layer Mobility Model

The doping-dependent and transverse field-dependent portion of the inversion and accumulation layer mobility model is selected by specifying the parameter `IALMob` in the command file. The complete model is obtained by combining this with the Hänsch model [10] for high-field saturation (see [Extended Canali Model on page 295](#)). For example:

```
Physics {
    Mobility (
        Enormal(IALMob)
        HighFieldSaturation(EparallelToInterface)
    )
}
```

In addition, parameter file specifications are required to use this model properly:

- The Hänsch model is selected by specifying the parameter $\alpha = 1$ in the `HighFieldDependence` section. In addition, the β exponent in the Hänsch model is equal to 2, which can be specified by setting $\beta_0 = 2$ and $\beta_{\text{exp}} = 0$.
- If no mobility doping dependency is specified in the command file, Sentaurus Device automatically combines `IALMob` with a constant mobility using Matthiessen's rule. Since `IALMob` already includes doping dependency, this must be effectively disabled by specifying a large value for μ_{\max} in the `ConstantMobility` section.

8: Mobility Models

Mobility Degradation at Interfaces

- Sentaurus Device omits the calculations of the transverse field-dependent mobility model if the normal field is very small. Since IALMob calculates both doping-dependent and transverse field-dependent mobility, it must be called for all values of the normal field. This can be accomplished by setting EnormMinimum = 0.0 in the IALMob section.

The required parameter file specifications are summarized here:

```
HighFieldDependence {
    alpha    = 1.0, 1.0      # [1]
    beta0    = 2.0, 2.0      # [1]
    betaexp  = 0.0, 0.0
}
ConstantMobility {
    mumax = 1.0e9, 1.0e9    # [cm^2/Vs]
}
IALMob {
    EnormMinimum = 0.0      # [V/cm]
}
```

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 271](#)). The inversion layer mobility in MOSFETs is degraded by Coulombic 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:

$$\frac{1}{\mu} = \frac{1}{\mu_{bsc}} + \frac{D}{\mu_{ac}} + \frac{D}{\mu_{sr}} \quad (235)$$

where $1/\mu_{bsc}$ is the contribution of Coulombic scattering, and $1/\mu_{ac}$, $1/\mu_{sr}$ 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 μ_{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):

$$\mu_{bsc}^{-1} = \mu_b^{-1} [D(1 + f_{sc}^\tau)^{1/\tau} + (1 - D)] \quad (236)$$

where μ_b is given by the bulk mobility model, and τ is a fit parameter. The screening function is given by:

$$f_{sc} = \left(\frac{N_1}{N_{A,0} + N_{D,0}} \right)^n \frac{N_{min}}{N_{A,0} + N_{D,0}} \quad (237)$$

where N_{min} is the minority carrier concentration.

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 [11]. The term μ_{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:

$$\mu_{ac} = C(T) \left(\frac{N_{A,0} + N_{D,0}}{N_2} \right)^a \frac{1}{F_\perp^\delta} \quad (238)$$

$$\mu_{sr} = B(T) \left(\frac{N_{A,0} + N_{D,0} + N_3}{N_4} \right)^b \frac{1}{F_\perp^\lambda} \quad (239)$$

F_\perp is the electric field normal to semiconductor–insulator interface, see [Computing Transverse Field on page 286](#).

The parameters for the model are accessible in the parameter set UniBoNormalDependence. [Table 47](#) lists the silicon default parameters. The reported parameters are detailed in the literature [12]. The model was calibrated with experiments [11][12] in the temperature range from 300 K to 700 K.

Table 47 Parameters of University of Bologna surface mobility model ($T_n = T/300\text{K}$)

Symbol	Parameter name	Electrons	Holes	Unit
N_1	N1	2.34×10^{16}	2.02×10^{16}	cm^{-3}
N_2	N2	4.0×10^{15}	7.8×10^{15}	cm^{-3}
N_3	N3	1.0×10^{17}	2.0×10^{15}	cm^{-3}
N_4	N3	2.4×10^{18}	6.6×10^{17}	cm^{-3}
B	B	$5.8 \times 10^1 T_n^{\gamma_B}$	$7.82 \times 10^{15} T_n^{\gamma_B}$	cm^2/Vs
γ_B	B_exp	0	1.4	1

8: Mobility Models

Mobility Degradation at Interfaces

Table 47 Parameters of University of Bologna surface mobility model ($T_n = T/300\text{K}$)

Symbol	Parameter name	Electrons	Holes	Unit
C	c	$1.86 \times 10^4 T_n^{-\gamma_c}$	$5.726 \times 10^3 T_n^{-\gamma_c}$	cm^2/Vs
γ_c	c_exp	2.1	1.3	1
τ	tau	1	3	1
η	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	1×10^{-6}	1×10^{-6}	cm
δ	delta	0.29	0.3	1
λ	lambda	2.64	2.24	1

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.

Furthermore, for vertices at the interface, an optional correction F_{corr} for the field value is available.

Normal to Interface

Assume that mobility degradation occurs at an interface Γ . By default, for a given point \vec{r} , Sentaurus Device locates the nearest vertex \vec{r}_i on the interface Γ , 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 . When the GeometricDistances option is specified in the Math section, the distance to the interface is the true geometric distance, and \hat{n} is the gradient of the distance to the interface. From \hat{n} , the normal electric field is:

$$F_\perp(\vec{r}) = |\vec{F}(\vec{r}) \cdot \hat{n} + F_{\text{corr}}| \quad (240)$$

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 Γ 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 explicitly in the Math section (see [Discretization Methods on page 93](#)).

In the following example, Sentaurus Device takes as Γ the union of interfaces between materials, OxideAsSemiconductor and Silicon, and regions, regionK1 and regionL1:

```
Math {
    ...
    EnormalInterface(regioninterface = ["regionK1/regionL1"],
                     materialinterface = ["OxideAsSemiconductor/Silicon"])
    ...
}
```

Normal to 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 $\vec{J}_c(\vec{r})$:

$$F_{c,\perp}(\vec{r}) = F(\vec{r}) \sqrt{1 - \left(\frac{\vec{F}(\vec{r}) \cdot \vec{J}_c(\vec{r})}{F(\vec{r}) J_c(\vec{r})} \right)^2} + F_{\text{corr}} \quad (241)$$

where $F_{n,\perp}$ is used for the evaluation of electron mobility and $F_{p,\perp}$ is used for the evaluation of hole mobility.

NOTE For very low current levels, the computation of the electric field component normal to the currents may 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:

$$F_{\text{corr}}(\vec{r}) = \frac{\alpha \rho(\vec{r})}{\epsilon} \quad (242)$$

where α is dimensionless and is specified with NormalFieldCorrection in the Math section (reasonable values range from 0 to 1; the default is zero), ρ is the space charge, ϵ is

8: Mobility Models

Carrier–Carrier Scattering

the dielectric constant in the semiconductor, and l is an estimate for the depth of the box of the vertex in the normal direction.

Carrier–Carrier Scattering

Two models are supported for the description of carrier–carrier scattering. One model is based on Choo [13] and Fletcher [14] and uses the Conwell–Weisskopf screening theory. As an alternative to the Conwell–Weisskopf model, Sentaurus Device supports the Brooks–Herring model. The carrier–carrier contribution to the overall mobility degradation is captured in the mobility term μ_{eh} . This is combined with the mobility contributions from other degradation models (μ_{other}) according to Matthiessen’s rule:

$$\frac{1}{\mu} = \frac{1}{\mu_{\text{other}}} + \frac{1}{\mu_{\text{eh}}} \quad (243)$$

Using Carrier–Carrier Scattering

The carrier–carrier scattering models are activated by specifying the `CarrierCarrierScattering` option to `Mobility`. 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

$$\mu_{\text{eh}} = \frac{D(T/300\text{K})^{3/2}}{\sqrt{np}} \left[\ln \left(1 + F \left(\frac{T}{300\text{K}} \right)^2 (pn)^{-1/3} \right) \right]^{-1} \quad (244)$$

The parameters D and F are accessible in the parameter file. The default values appropriate for silicon are given in [Table 48](#).

Brooks–Herring Model

$$\mu_{\text{eh}} = \frac{c_1(T/300\text{ K})^{3/2}}{\sqrt{np}} \frac{1}{\phi(\eta_0)} \quad (245)$$

where $\phi(\eta_0) = \ln(1 + \eta_0) - \eta_0/(1 + \eta_0)$ and:

$$\eta_0(T) = \frac{c_2}{N_C F_{-1/2}(n/N_C) + N_V F_{-1/2}(p/N_V)} \left(\frac{T}{300\text{ K}} \right)^2 \quad (246)$$

Table 49 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 48 Conwell–Weisskopf model: Default parameters

Symbol	Parameter name	Value	Unit
D	D	1.04×10^{21}	$\text{cm}^{-1}\text{V}^{-1}\text{s}^{-1}$
F	F	7.452×10^{13}	cm^{-2}

Table 49 Brooks-Herring model: Default parameters

Symbol	Parameter name	Value	Unit
c_1	c1	1.56×10^{21}	$\text{cm}^{-1}\text{V}^{-1}\text{s}^{-1}$
c_2	c2	7.63×10^{19}	cm^{-3}

Philips Unified Mobility Model

The Philips unified mobility model, proposed by Klaassen [15], 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.

8: Mobility Models

Philips Unified Mobility Model

Using Philips Model

The Philips unified mobility model is activated by specifying the `PhuMob` option to `Mobility`:

```
Physics{ Mobility( PhuMob ... ) ... }
```

The model can also be activated with one or two additional options:

```
Physics{ Mobility( PhuMob( [ Arsenic | Phosphorus ]  
[ Klaassen | Meyer ] ) ... ) ... }
```

The option `Arsenic` or `Phosphorus` specifies which parameters (see [Table 51 on page 293](#)) are used. These parameters reflect the different electron mobility degradation that is observed in the presence of these donor species.

The option `Klaassen` or `Meyer` specifies which parameters (see [Table 50 on page 293](#)) are used for the evaluation of $G(P_i)$. Sentaurus Device defaults to the `Klaassen` parameters.

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 Model

Sentaurus Device provides an extended PMI reimplementation of the Philips unified mobility model (see [Doping-dependent Mobility on page 942](#)). To use it, specify `PhuMob2` as the option for `DopingDependence`:

```
Physics{ Mobility(DopingDependence(PhuMob2) ...) ... }
```

The reimplementation uses the `PhuMob2` parameter set. The new parameter, `FACT_G`, has a default value of 1. It modifies the function $G(P_i)$ shown in [Eq. 259](#) by a factor $G(P_i) = \text{FACT_G}$ (right-hand side in [Eq. 259](#)). Otherwise, the reimplementation fully agrees with the model described below.

Philips Model Description

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:

$$\frac{1}{\mu_{i,b}} = \frac{1}{\mu_{i,L}} + \frac{1}{\mu_{i,DAeh}} \quad (247)$$

In Eq. 247 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:

$$\mu_{i,L} = \mu_{i,max} \left(\frac{T}{300\text{ K}} \right)^{-\theta_i} \quad (248)$$

The second contribution has the form:

$$\mu_{i,DAeh} = \mu_{i,N} \left(\frac{N_{i,sc}}{N_{i,sc,eff}} \right) \left(\frac{N_{i,ref}}{N_{i,sc}} \right)^{\alpha_i} + \mu_{i,c} \left(\frac{n+p}{N_{i,sc,eff}} \right) \quad (249)$$

with:

$$\mu_{i,N} = \frac{\mu_{i,max}^2}{\mu_{i,max} - \mu_{i,min}} \left(\frac{T}{300\text{ K}} \right)^{3\alpha_i - 1.5} \quad (250)$$

$$\mu_{i,c} = \frac{\mu_{i,max} \mu_{i,min}}{\mu_{i,max} - \mu_{i,min}} \left(\frac{300K}{T} \right)^{0.5} \quad (251)$$

for the electrons:

$$N_{e,sc} = N_D^* + N_A^* + p \quad (252)$$

$$N_{e,sc,eff} = N_D^* + G(P_e)N_A^* + f_e \frac{p}{F(P_e)} \quad (253)$$

and for holes:

$$N_{h,sc} = N_A^* + N_D^* + n \quad (254)$$

$$N_{h,sc,eff} = N_A^* + G(P_h)N_D^* + f_h \frac{n}{F(P_h)} \quad (255)$$

8: Mobility Models

Philips Unified Mobility Model

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:

$$N_D^* = N_{D,0}Z_D = N_{D,0}\left[1 + \frac{N_{D,0}^2}{c_D N_{D,0}^2 + N_{D,\text{ref}}^2}\right] \quad (256)$$

$$N_A^* = N_{A,0}Z_A = N_{A,0}\left[1 + \frac{N_{A,0}^2}{c_A N_{A,0}^2 + N_{A,\text{ref}}^2}\right] \quad (257)$$

The analytic functions $G(P_i)$ and $F(P_i)$ in Eq. 253 and Eq. 255 describe minority impurity and electron–hole scattering. They are given by:

$$F(P_i) = \frac{0.7643 P_i^{0.6478} + 2.2999 + 6.5502(m^*_i/m^*_j)}{P_i^{0.6478} + 2.3670 - 0.8552(m^*_i/m^*_j)} \quad (258)$$

and:

$$G(P_i) = 1 - a_g \left[b_g + P_i \left(\frac{m_0}{m^*_i 300 \text{K}} \frac{T}{300 \text{K}} \right)^{\alpha_g} \right]^{-\beta_g} + c_g \left[P_i \left(\frac{m^*_i 300 \text{K}}{m_0} \frac{T}{300 \text{K}} \right)^{\alpha'_g} \right]^{-\gamma_g} \quad (259)$$

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:

$$P_i = \left[\frac{f_{\text{CW}}}{3.97 \times 10^{13} \text{cm}^{-2} N_{i,\text{sc}}^{-2/3}} + f_{\text{BH}} \frac{(n+p)}{1.36 \times 10^{20} \text{cm}^{-3} m^*_i} \frac{m_0}{m^*_i} \right]^{-1} \left(\frac{T}{300 \text{K}} \right)^2 \quad (260)$$

If the Klaassen parameter set is selected, 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 the Meyer parameter set is selected, Eq. 259 is used independently of the value of P_i .

Philips Model Parameters

Sentaurus Device supports two different built-in values for each of the parameters a_g , b_g , c_g , α_g , α'_g , β_g , and γ_g in [Eq. 259](#). They are selected by the Klaassen or Meyer option (see [Using Philips Model on page 290](#)). The corresponding values are given in [Table 50](#). Sentaurus Device defaults to the Klaassen parameters.

Table 50 Philips unified mobility model parameters: Klaassen versus Meyer

Symbol	Klaassen	Meyer	Unit
a_g	0.89233	4.41804	1
b_g	0.41372	39.9014	1
c_g	0.005978	0.52896	1
α_g	0.28227	0.0001	1
α'_g	0.72169	1.595187	1
β_g	0.19778	0.38297	1
γ_g	1.80618	0.25948	1

Other parameters for the Philips unified mobility model are accessible in the parameter set PhuMob.

[Table 51](#) and [Table 52 on page 294](#) list the silicon defaults for other parameters. Sentaurus Device supports different parameters for electron mobility, which are optimized for situations where the dominant donor species in the silicon is either arsenic or phosphorus. The arsenic parameters are used by default.

Table 51 Philips unified mobility model parameters (silicon), group 1

Symbol	Parameter name	Electrons (arsenic)	Electrons (phosphorus)	Holes (boron)	Unit
μ_{\max}	mumax_*	1417	1414	470.5	cm^2/Vs
μ_{\min}	mumin_*	52.2	68.5	44.9	cm^2/Vs
θ	theta_*	2.285	2.285	2.247	1
$N_{\{\text{e},\text{h}\},\text{ref}}$	n_ref_*	9.68×10^{16}	9.2×10^{16}	2.23×10^{17}	cm^{-3}
α	alpha_*	0.68	0.711	0.719	1

8: Mobility Models

High-Field Saturation

Table 52 Philips unified mobility model parameters (silicon), group 2

Symbol	Parameter name	Donor (*=D)	Acceptor (*=A)	Unit
$N_{\{D,A\},ref}$	nref_*	4×10^{20}	7.2×10^{20}	cm^{-3}
c	cref_*	0.21	0.5	1

The original Philips unified mobility model uses four fit parameters: the weight factors f_{CW} and f_{BH} , and the ‘effective masses’ m^*_e and m^*_h . The optimal parameter set, determined by accurate fitting to experimental data [15] is shown in [Table 53](#). 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 298](#)).

Table 53 Philips unified mobility model parameters (silicon), group 3

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_{CW}	f_CW	2.459	1
f_{BH}	f_BH	3.828	1
f_e	f_e	1.0	1
f_h	f_h	1.0	1

High-Field Saturation

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 and the transferred electron model 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 568](#).

Using High-Field Saturation

The high-field saturation models comprise three submodels: the actual mobility model, the velocity saturation model, and the driving force model. With a 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 295](#)).

The flag TransferredElectronEffect selects the transferred electron model (see [Transferred Electron Model on page 297](#)).

The flags CarrierTempDriveBasic and CarrierTempDriveME activate the ‘basic’ and the Meinerzhagen–Engl model, respectively (see [Basic Model on page 297](#) and [Meinerzhagen–Engl Model on page 298](#)). These two models require hydrodynamic simulations.

For the Canali model and the transferred electron model, the driving force model is selected by a flag to eHighFieldSaturation or hHighFieldSaturation. Available flags are GradQuasiFermi (the default), Eparallel, EparallelToInterface, and CarrierTempDrive (see [Driving Force Models on page 300](#)). The latter is only available in hydrodynamic simulations. 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, a velocity saturation model can be selected in the HighFieldDependence parameter set (see [Velocity Saturation Models on page 299](#)).

Extended Canali Model

The Canali model [16] originates from the Caughey–Thomas formula [17], but has temperature-dependent parameters, which were fitted up to 430 K by Canali *et al.* [16]:

$$\mu(F) = \frac{(\alpha + 1)\mu_{\text{low}}}{\alpha + \left[1 + \left(\frac{(\alpha + 1)\mu_{\text{low}} F_{\text{hfs}}}{v_{\text{sat}}} \right)^{\beta} \right]^{1/\beta}} \quad (261)$$

where μ_{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 268](#) to [Philips Unified Mobility Model on page 289](#)). The exponent β is temperature dependent according to:

$$\beta = \beta_0 \left(\frac{T}{300 \text{ K}} \right)^{\beta_{\text{exp}}} \quad (262)$$

Details about the saturation velocity v_{sat} and driving field F_{hfs} are discussed in [Velocity Saturation Models on page 299](#) and [Driving Force Models on page 300](#). All other parameters are accessible in the parameter set HighFieldDependence.

The silicon default values are listed in [Table 54 on page 296](#).

A modified version of the Canali model is the Hänsch model [10]. It is activated by setting the parameter $\alpha = 1$. The Hänsch model is part of the Lucent mobility model (see [Lucent Model on page 298](#)). When using the hydrodynamic driving force [Eq. 273](#), α must be zero.

For the hydrodynamic driving force, [Eq. 273](#) can be substituted into [Eq. 261](#). Solving for μ yields the hydrodynamic Canali model:

$$\mu = \frac{\mu_{\text{low}}}{\left[\sqrt{1 + \gamma^2 \max(w_c - w_0, 0)^\beta} + \gamma \max(w_c - w_0, 0)^{\beta/2} \right]^{2/\beta}} \quad (263)$$

where γ is given by:

$$\gamma = \frac{1}{2} \left(\frac{\mu_{\text{low}}}{q \tau_{e,c} v_{\text{sat}}^2} \right)^{\beta/2} \quad (264)$$

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 μ_{low} and the mobility given in [Eq. 263](#). K_{dT} defaults to 0.2 and can be accessed in the parameter set `HighFieldDependence`.

Table 54 Canali model parameters (default values for silicon)

Symbol	Parameter name	Electrons	Holes	Unit
β_0	beta0	1.109	1.213	1
β_{exp}	betaexp	0.66	0.17	1
α	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 [18]:

$$\mu = \frac{\mu_{\text{low}} + \left(\frac{v_{\text{sat}}}{F_{\text{hfs}}} \right) \left(\frac{F_{\text{hfs}}}{E_0} \right)^4}{1 + \left(\frac{F_{\text{hfs}}}{E_0} \right)^4} \quad (265)$$

Details of the saturation velocity v_{sat} and the driving field F_{hfs} are discussed in [Velocity Saturation Models on page 299](#) and [Driving Force Models on page 300](#). The reference field strength E_0 can be set in the parameter set `HighFieldDependence`.

The `HighFieldDependence` parameter set also includes a variable K_{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_{vmax} is the field strength at which the velocity is at its maximum, $v_{\text{max}} = \mu F_{\text{vmax}}$. In this interval, Eq. 265 is replaced by a polynomial that produces the same values and derivatives at the points F_{vmax} and $K_{\text{smooth}} F_{\text{vmax}}$. It is sometimes numerically advantageous to set $K_{\text{smooth}} \approx 20$.

Table 55 Transferred electron model: Default parameters

Symbol	Parameter	Electrons	Holes	Unit
E_0	<code>E0_TrEf</code>	4000	4000	Vcm^{-1}
K_{smooth}	<code>Ksmooth_TrEf</code>	1	1	1

Basic Model

According to this very simple model, the mobility decays inversely with the carrier temperature:

$$\mu = \mu_{\text{low}} \left(\frac{300 \text{ K}}{T_c} \right) \quad (266)$$

where μ_{low} is the low-field mobility and T_c is the carrier temperature.

Meinerzhagen–Engl Model

According to the Meinerzhagen–Engl model [19], the high field mobility degradation is given by:

$$\mu = \frac{\mu_{\text{low}}}{\left[1 + \left(\mu_{\text{low}} \frac{(W_c - W_0)}{q\tau_{e,c} v_{\text{sat}}^2} \right)^\beta \right]^{1/\beta}} \quad (267)$$

where $\tau_{e,c}$ is the energy relaxation time. The coefficients of the saturation velocity v_{sat} (see Eq. 268) and the exponent β (see Eq. 262) 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 56 and Table 57 on page 299.

Table 56 Meinerzhagen–Engl model: Default parameters

Silicon	Electrons	Holes	Unit
β_0	0.6	0.6	1
β_{exp}	0.01	0.01	1

Lucent Model

The Lucent model has been developed by Darwish *et al.* [6]. Sentaurus Device implements this model as a combination of:

- An extended Philips unified mobility model (see [Philips Unified Mobility Model on page 289](#)) with the parameters $f_e = 0$ and $f_h = 0$ (see [Table 53 on page 294](#)). The Lucent model described in [6] 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 274](#)) with the parameters from [Table 40 on page 276](#).

- The Hänsch model (see [Extended Canali Model on page 295](#)) with the parameter $\alpha = 1$ (see [Table 54 on page 296](#)). In addition, the β 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 two velocity saturation models. Model 1 is part of the Canali model and is given by:

$$v_{\text{sat}} = v_{\text{sat},0} \left(\frac{300 \text{ K}}{T} \right)^{v_{\text{sat},\text{exp}}} \quad (268)$$

This model is recommended for silicon.

Model 2 is recommended for GaAs. Here, v_{sat} is given by:

$$v_{\text{sat}} = \begin{cases} A_{\text{vsat}} - B_{\text{vsat}} \left(\frac{T}{300 \text{ K}} \right) & v_{\text{vsat}} > v_{\text{sat,min}} \\ v_{\text{sat,min}} & \text{otherwise} \end{cases} \quad (269)$$

The 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, [Eq. 268](#) is used. If `Vsat_formula` is set to 2, [Eq. 269](#) 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 57 Velocity saturation parameters

Symbol	Parameter	Electrons	Holes	Unit
$v_{\text{sat},0}$	<code>vsat0</code>	1.07×10^7	8.37×10^6	cm/s
$v_{\text{sat},\text{exp}}$	<code>vsatexp</code>	0.87	0.52	1
A_{vsat}	<code>A_vsat</code>	1.07×10^7	8.37×10^6	cm/s
B_{vsat}	<code>B_vsat</code>	3.6×10^6	3.6×10^6	cm/s
$v_{\text{sat,min}}$	<code>vsat_min</code>	5.0×10^5	5.0×10^5	cm/s

Driving Force Models

Sentaurus Device supports four different models for the driving force F_{hfs} . For the first model (flag `Eparallel`), the driving field for electrons is the electric field parallel to the electron current:

$$F_{\text{hfs}, n} = \vec{F} \cdot \frac{\vec{J}_n}{J_n} \quad (270)$$

For the second model (flag `GradQuasiFermi`), the driving field for electrons is:

$$F_{\text{hfs}, n} = |\nabla \Phi_n| \quad (271)$$

The driving fields for holes are analogous.

NOTE Usually, [Eq. 270](#) and [Eq. 271](#) give the same or very similar results. However, numerically, one model may prove to be more stable. For example, in regions with small current, the evaluation of the parallel electric field can be numerically problematic.

For numeric reasons, Sentaurus Device actually implements generalizations of [Eq. 270](#) and [Eq. 271](#), where $F_{\text{hfs}, n} = n \vec{F} \cdot \vec{J}_n / J_n (n + n_0)$ and $F_{\text{hfs}, n} = n |\nabla \Phi_n| / (n + n_0)$, respectively. Here, n_0 is a numeric damping parameter. The values for electrons and holes default to zero, and are set (in cm^{-3}) with the `eDrForceRefDens` and `hDrForceRefDens` parameters in the Math section. Using positive values for n_0 can improve convergence for problems where strong generation–recombination occurs in regions with small density.

The third model (keyword `EparallelToInterface`) computes the driving force as the electric field parallel to the closest semiconductor–insulator interface:

$$F_{\text{hfs}} = |(I - \hat{n}\hat{n}^T)\vec{F}| \quad (272)$$

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 Interface on page 286](#)).

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 model will only give valid results if the current flows predominantly parallel to the interface (such as in the channel of MOSFET devices).

The fourth model (keyword `CarrierTempDrive`) requires hydrodynamic simulation. The driving field for electrons is:

$$F_{\text{hfs},n} = \sqrt{\frac{\max(w_n - w_0, 0)}{\tau_{e,n} q \mu_n}} \quad (273)$$

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.

Monte Carlo-computed Mobility for Strained Silicon

Based on Monte Carlo simulations [20], a parameter file of Sentaurus Device, `StrainedSilicon.par`, is created in the library of materials (see [Library of Materials on page 117](#)). This file contains in-plane transport parameters at 300 K for silicon under biaxial tensile strain that is present when a thin silicon film is grown on top of a relaxed `SiliconGermanium` substrate. (*In-plane* refers to charge transport that is parallel to the interface to `SiliconGermanium`, as is the case in MOSFETs.)

In the Physics section of the command file, the germanium content (`XFraction`) of the `SiliconGermanium` substrate at the interface to the `StrainedSilicon` channel must be specified (this value determines the strain in the top silicon film) according to:

```
MoleFraction (RegionName= ["TopLayer"] XFraction = 0.2 Grading = 0.0)
```

where the material of `TopLayer` is `StrainedSilicon`.

Band offsets and bulk mobility data are obtained from the model-solid theory of Van de Walle and descriptions of Monte Carlo simulations [20].

A parameterization of the surface mobility model as a function of strain is not yet possible. The present values for the parameters B and C of the surface mobility were extracted in a 1 μm bulk MOSFET at a high effective field (where the silicon control measurements of the references [21][22] were in good agreement with the universal mobility curve of silicon, and where the neglected effect of the SiGe substrate is minimal) of approximately 0.7 MV/cm to reproduce reported experimental data [21][22], for the typical germanium content in the `SiliconGermanium` substrate of 30%. The values for other germanium contents are given as comments.

8: Mobility Models

Monte Carlo–computed Mobility for Strained SiGe in npn-SiGe HBTs

Monte Carlo–computed Mobility for Strained SiGe in npn-SiGe HBTs

Based on Monte Carlo simulations [23], a Sentaurus Device parameter file `SiGeHBT.par` has been created in the library of materials (see [Library of Materials on page 117](#)). This file contains transport parameters at 300 K for silicon germanium under biaxial compressive strain present when a thin SiGe film is grown on top of a relaxed silicon substrate, which occurs in the base of npn-SiGe heterojunction bipolar transistors (HBTs).

The electron parameters refer to the out-of-plane direction (that is, perpendicular to the SiGe–silicon interface) and the hole parameters to the in-plane direction (that is, parallel to the SiGe–silicon interface). The profile of the germanium content can either originate from a process simulation or be specified in the command file of Sentaurus Device (see [Mole-Fraction Specification on page 550](#)).

The transport parameters have been obtained from the full-band Monte Carlo simulations [23].

The band gap in relaxed SiGe alloys has been extracted from the measurements in [24] and the values in strained SiGe calculated according to the model solid theory of C. G. Van de Walle.

Consistent limiting values for silicon (and polysilicon) are also provided.

Incomplete Ionization–dependent Mobility Models

Sentaurus Device supports incomplete ionization–dependent mobility models. This dependence is activated by specifying the keyword `IncompleteIonization` in `Mobility` sections. The incomplete ionization model (see [Chapter 6 on page 245](#)) must be activated also. The `Physics` sections 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_{tot} , Sentaurus Device will use N_A , N_D , N_i .

The following mobility models depend on incomplete ionization:

- Masetti model, see [Eq. 193](#)
- Arora model, see [Eq. 194](#)
- University of Bologna bulk model, see [Eq. 198–Eq. 200](#)

- Lombardi model, see [Eq. 201](#) and [Eq. 204](#)
- University of Bologna inversion layer model, see [Eq. 237](#)–[Eq. 239](#)
- Philips unified model, see [Eq. 256](#) and [Eq. 257](#)

Poole–Frenkel Mobility (Organic Material 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. The mobility as a function of the electric field is given by:

$$\mu = \mu_0 \exp\left(-\frac{E_0}{kT}\right) \exp\left(\sqrt{F}\left(\frac{\beta}{T} - \gamma\right)\right) \quad (274)$$

where μ_0 is the low-field mobility, β and γ are fitting parameters, E_0 is the effective activation energy, and F is the driving force (electric field).

The parameters E_0 , β , and γ 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 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 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 value is 0.1.

8: Mobility Models

Mobility Averaging

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)]) ... }
```

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 274](#)) 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.

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, it disables the mobility dependency on IncompleteIonization if this mobility option is specified.
- The 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")
```

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.
- [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. 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.

8: Mobility Models

References

- [7] 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.
- [8] W. J. Zhu and T. P. Ma, “Temperature Dependence of Channel Mobility in HfO₂-Gated NMOSFETs,” *IEEE Electron Device Letters*, vol. 25, no. 2, pp. 89–91, 2004.
- [9] S. A. Mujtaba, *Advanced Mobility Models for Design and Simulation of Deep Submicrometer MOSFETs*, Ph.D. thesis, Stanford University, Stanford, CA, USA, December 1995.
- [10] 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.
- [11] 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.
- [12] G. Baccarani, *A Unified mobility model for Numerical Simulation, Parasitics Report*, DEIS-University of Bologna, Bologna, Italy, 1999.
- [13] 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.
- [14] N. H. Fletcher, “The High Current Limit for Semiconductor Junction Devices,” *Proceedings of the IRE*, vol. 45, no. 6, pp. 862–872, 1957.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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.

- [21] 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.
- [22] 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.
- [23] F. M. Bufler, *Full-Band Monte Carlo Simulation of Electrons and Holes in Strained Si and SiGe*, München: Herbert Utz Verlag, 1998.
- [24] 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.

8: Mobility Models

References

Generation–Recombination

This chapter describes the generation–recombination processes that can be modeled in Sentaurus Device.

Generation–recombination processes are processes that 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 exception is one of the band-to-band tunneling models, see [Dynamic Nonlocal Path Band-to-Band Model on page 339](#). For other models that couple the conduction and valence bands and account for spatial transport of charge, see [Tunneling and Traps on page 357](#) and [Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 518](#).

Shockley–Read–Hall Recombination

Recombination through deep defect levels in the gap is usually labeled Shockley–Read–Hall (SRH) recombination. In Sentaurus Device, the following form is implemented:

$$R_{\text{net}}^{\text{SRH}} = \frac{np - n_{\text{i,eff}}^2}{\tau_p(n + n_1) + \tau_n(p + p_1)} \quad (275)$$

with:

$$n_1 = n_{\text{i,eff}} \exp\left(\frac{E_{\text{trap}}}{kT}\right) \quad (276)$$

and:

$$p_1 = n_{\text{i,eff}} \exp\left(\frac{-E_{\text{trap}}}{kT}\right) \quad (277)$$

where E_{trap} is the difference between the defect level and intrinsic level. The variable E_{trap} is accessible in the parameter file. The silicon default value is $E_{\text{trap}} = 0$.

9: Generation–Recombination

Shockley–Read–Hall Recombination

The minority lifetimes τ_n and τ_p are modeled as a product of a doping-dependent (see [SRH Doping Dependence on page 311](#)), field-dependent (see [SRH Field Enhancement on page 313](#)), and temperature-dependent (see [SRH Temperature Dependence on page 312](#)) factor:

$$\tau_c = \tau_{\text{dop}} \frac{f(T)}{1 + g_c(F)}, c = n, p \quad (278)$$

where $c = n$ or $c = p$ for holes. For an additional density dependency of the lifetimes, see [Trap-assisted Auger Recombination on page 318](#).

For simulations that use Fermi statistics (see [Fermi Statistics on page 202](#)) or quantization (see [Chapter 7 on page 251](#)), Eq. 275 needs to be generalized. The modified equation reads:

$$R_{\text{net}}^{\text{SRH}} = \frac{np - \gamma_n \gamma_p n_{\text{i,eff}}^2}{\tau_p(n + \gamma_n n_1) + \tau_n(p + \gamma_p p_1)} \quad (279)$$

where γ_n and γ_p are given by [Eq. 90, p. 202](#) and [Eq. 91, p. 202](#).

Using SRH Recombination

The 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

The doping dependence of the SRH lifetimes is modeled in Sentaurus Device with the Scharfetter relation:

$$\tau_{\text{dop}}(N_{A,0} + N_{D,0}) = \tau_{\min} + \frac{\tau_{\max} - \tau_{\min}}{1 + \left(\frac{N_{A,0} + N_{D,0}}{N_{\text{ref}}}\right)^{\gamma}} \quad (280)$$

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]. Default values are listed in [Table 58 on page 313](#). The Scharfetter relation is used when the argument `DopingDependence` is specified for the SRH recombination. Otherwise, $\tau = \tau_{\max}$ is used.

The evaluation of the SRH lifetimes according to the Scharfetter model is activated by specifying the additional argument `DopingDependence` for the `SRH` keyword in the `Recombination` statement:

```
Physics{ Recombination( SRH( DopingDependence ... ) ... ) ... }
```

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 (refer to the *Sentaurus Structure Editor User Guide*).

The names of the datasets for the 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 τ_{\max} in [Eq. 280](#).

The lifetime data can be in a separate file from the doping, but must correspond to the same grid.

SRH Temperature Dependence

To date, there is no consensus on the temperature dependence of the SRH lifetimes. This appears to originate from a different understanding of lifetime. From measurements of the recombination lifetime [5][6]:

$$\tau = \delta n / R \quad (281)$$

in power devices (δn 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 [5][6]:

$$\tau(T) = \tau_0 \left(\frac{T}{300 \text{ K}} \right)^\alpha \quad (282)$$

or an exponential expression of the form:

$$\tau(T) = \tau_0 e^{c \left(\frac{T}{300 \text{ K}} - 1 \right)} \quad (283)$$

A calculation using the low-temperature approximation of multiphonon theory [7] gives:

$$\tau_{\text{SRH}}(T) = \tau_{\text{SRH}}(300 \text{ K}) \cdot f(T) \quad \text{with} \quad f(T) = \left(\frac{T}{300 \text{ K}} \right)^{T_\alpha} \quad (284)$$

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 $\tau_{\text{SRH}}(T)$.

In Sentaurus Device, the power law model, Eq. 282, can be activated with the keyword TempDependence in the SRH statement:

```
Physics{ Recombination( SRH( TempDependence ... ) ... ) }
```

Additionally, Sentaurus Device supports an exponential model for $f(T)$:

$$f(T) = e^{c \left(\frac{T}{300 \text{ K}} - 1 \right)} \quad (285)$$

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 parameter set `Scharfetter`.

Table 58 Default parameters for doping- and temperature-dependent SRH lifetime

Symbol	Parameter name	Electrons	Holes	Unit
τ_{\min}	<code>taumin</code>	0	0	s
τ_{\max}	<code>taumax</code>	1×10^{-5}	3×10^{-6}	s
N_{ref}	<code>Nref</code>	1×10^{16}	1×10^{16}	cm^{-3}
γ	<code>gamma</code>	1	1	1
T_{α}	<code>Talpha</code>	-1.5	-1.5	1
C	<code>Tcoeff</code>	2.55	2.55	1
E_{trap}	<code>Etrap</code>	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 3×10^5 V/cm in certain regions of the device. For example, the I – V characteristics of reverse-biased p–n 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 two field-enhancement models: the Schenk trap-assisted tunneling model (see [Schenk Trap-assisted Tunneling \(TAT\) Model on page 314](#)) and the Hurkx trap-assisted tunneling model (see [Hurkx TAT Model on page 316](#)). The Hurkx model is also available for trap capture and emission rates (see [Hurkx Model for Cross Sections on page 355](#)).

Using Field Enhancement

The local field-dependence of the SRH lifetimes is activated by parameters in the `ElectricField` option of `SRH`:

```
SRH( ...
    ElectricField (
        Lifetime = Schenk | Hurkx | Constant
        DensityCorrection = Local | None
    )
)
```

`Lifetime` selects the lifetime model. The default is `Constant`, for field-independent lifetime. `Schenk` selects the Schenk lifetimes (see [Schenk Trap-assisted Tunneling \(TAT\) Model on page 314](#)), and `Hurkx` selects the Hurkx lifetimes (see [Hurkx TAT Model on page 316](#)).

`DensityCorrection` defaults to `None`. A value of `Local` selects the model described in [Schenk TAT Density Correction on page 316](#).

For backward compatibility:

- `SRH` (Tunneling) selects `Lifetime = Schenk` and `DensityCorrection = Local`.
- `SRH(Tunneling(Hurkx))` selects `Lifetime = Hurkx` and `DensityCorrection = None`.

NOTE The inclusion of defect-assisted tunneling may lead to convergence problems. In such cases, it is recommended that the flag `NoSRHperPotential` is specified in the `Math` section. This causes Sentaurus Device to exclude derivatives of $g(F)$ with respect to the potential from the Jacobian matrix.

Schenk Trap-assisted Tunneling (TAT) Model

The field dependence of the recombination rate is taken into account by the field enhancement factors:

$$[1 + g(F)]^{-1} \quad (286)$$

of the SRH lifetimes [7] (see [Eq. 278](#)).

In the case of electrons, $g(F)$ has the form:

$$g_n(F) = \left(1 + \frac{(\hbar\Theta)^{3/2} \sqrt{E_t - E_0}}{E_0 \hbar\omega_0} \right)^{-\frac{1}{2}} \frac{(\hbar\Theta)^{3/4} (E_t - E_0)^{1/4}}{2 \sqrt{E_t E_0}} \left(\frac{\hbar\Theta}{kT} \right)^{\frac{3}{2}} \times \exp \left(-\frac{E_t - E_0}{\hbar\omega_0} + \frac{\hbar\omega_0 - kT}{2\hbar\omega_0} + \frac{2E_t + kT}{2\hbar\omega_0} \ln \frac{E_t}{\varepsilon_R} - \frac{E_0}{\hbar\omega_0} \ln \frac{E_0}{\varepsilon_R} + \frac{E_t - E_0}{kT} - \frac{4}{3} \left(\frac{E_t - E_0}{\hbar\Theta} \right)^2 \right) \quad (287)$$

where E_0 denotes the energy of an optimum horizontal transition path, which depends on field strength and temperature in the following way:

$$E_0 = 2\sqrt{\varepsilon_F} [\sqrt{\varepsilon_F + E_t + \varepsilon_R} - \sqrt{\varepsilon_F}] - \varepsilon_R, \quad \varepsilon_F = \frac{(2\varepsilon_R kT)^2}{(\hbar\Theta)^3} \quad (288)$$

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 Eq. 287 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_{trap} of Eq. 276 and Eq. 277 by:

$$E_t = \frac{1}{2} E_{g,\text{eff}} + \frac{3}{4} kT \ln \left(\frac{m_n}{m_p} \right) - E_{\text{trap}} - (32R_C \hbar^3 \Theta^3)^{1/4} \quad (289)$$

where the effective Rydberg constant R_C is:

$$R_C = m_C \left(\frac{Z^2}{\varepsilon^2} \right) Ry \quad (290)$$

where Ry is the Rydberg energy (13.606 eV), ε is the relative dielectric constant, and Z is a fit parameter.

For holes, E_t is given by:

$$E_t = \frac{1}{2} E_{g,\text{eff}} - \frac{3}{4} kT \ln \left(\frac{m_n}{m_p} \right) + E_{\text{trap}} - (32R_V \hbar^3 \Theta^3)^{1/4} \quad (291)$$

Note that E_{trap} is measured from the intrinsic level and not from mid gap. The zero-field lifetime τ_{SRH} is defined by Eq. 280.

Schenk TAT Density Correction

In the original Schenk model [7], the density n in Eq. 275 is replaced by:

$$\tilde{n} = n \exp\left(-\frac{\gamma_n |\nabla E_{F,n}| (E_t - E_0)}{kTF}\right) \quad (292)$$

with E_0 and E_t according to Eq. 288 and Eq. 289. 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_{ref} and p_{ref} are specified (in cm^{-3}) by the DenCorRef parameter pair in the TrapAssistedTunneling parameter set. They default to 10^3 cm^{-3} .

Hurkx TAT Model

The following equations apply to electrons and holes. The lifetimes and capture cross sections become functions of the trap-assisted tunneling factor $g(F) = \Gamma_{\text{tat}}$:

$$\tau = \tau_0 / (1 + \Gamma_{\text{tat}}), \quad \sigma = \sigma_0 (1 + \Gamma_{\text{tat}}) \quad (293)$$

where Γ_{tat} is given by:

$$\Gamma_{\text{tat}} = \int_0^{E_n} \exp\left[u - \frac{2}{3} \frac{\sqrt{u^3}}{\tilde{E}}\right] du \quad (294)$$

with the approximate solutions:

$$\Gamma_{\text{tat}} \approx \begin{cases} \sqrt{\pi} \tilde{E} \cdot \exp\left[\frac{1}{3} \tilde{E}^2\right] \left(2 - \operatorname{erfc}\left[\frac{1}{2} \left(\frac{\tilde{E}_n}{\tilde{E}} - \tilde{E}\right)\right]\right), & \tilde{E} \leq \sqrt{\tilde{E}_n} \\ \sqrt{\pi} \tilde{E} \cdot \tilde{E}_n^{1/4} \exp\left[-\tilde{E}_n + \tilde{E} \sqrt{\tilde{E}_n} + \frac{1}{3} \tilde{E} \sqrt{\tilde{E}_n^3}\right] \operatorname{erfc}\left[\tilde{E}_n^{1/4} \sqrt{\tilde{E}} - \tilde{E}_n^{3/4} / \sqrt{\tilde{E}}\right], & \tilde{E} > \sqrt{\tilde{E}_n} \end{cases} \quad (295)$$

where \tilde{E} and \tilde{E}_n are respectively defined as:

$$\tilde{E} = \frac{E}{E_0}, \text{ where } E_0 = \frac{\sqrt{8m_0 m_t k^3 T^3}}{qh} \quad (296)$$

$$\tilde{E}_n = \frac{E_n}{kT} = \begin{cases} 0, & kT \ln \frac{n}{n_i} > 0.5E_g \\ \frac{0.5E_g}{k_B T} - \ln \frac{n}{n_i}, & E_{\text{trap}} \leq kT \ln \frac{n}{n_i} \leq 0.5E_g \\ \frac{0.5E_g}{kT} - E_{\text{trap}}, & E_{\text{trap}} > kT \ln \frac{n}{n_i} \end{cases} \quad (297)$$

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 (τ) or from trap equations if it is applied to cross sections (σ).

When quantization is active (see [Chapter 7 on page 251](#)), the classical density:

$$n_{\text{cl}} = n \exp\left(\frac{\Lambda}{kT}\right) \quad (298)$$

rather than the true density n enters [Eq. 297](#).

Field-Enhancement Parameters

The parameters for the Schenk model are accessible in the parameter set `TrapAssistedTunneling`. 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 Vcm^{-1}) used for smoothing at small electric fields. A value of zero (the default) disables smoothing.

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>
}
```

9: Generation–Recombination

Shockley–Read–Hall Recombination

Trap-assisted Auger Recombination

Trap-assisted Auger (TAA) recombination is a modification to the SRH recombination (see [Shockley–Read–Hall Recombination on page 309](#)) and coupled defect level (see [Coupled Defect Level \(CDL\) Recombination on page 320](#)) models. When TAA is active, Sentaurus Device uses the lifetimes [4]:

$$\frac{\tau_p}{1 + \tau_p/\tau_p^{\text{TAA}}} \quad (299)$$

$$\frac{\tau_n}{1 + \tau_n/\tau_n^{\text{TAA}}} \quad (300)$$

in place of the lifetimes τ_p and τ_n in [Eq. 275](#) and [Eq. 307](#).

The TAA lifetimes in [Eq. 299](#) depend on the carrier densities:

$$\frac{1}{\tau_n^{\text{TAA}}} \approx C_p^{\text{TAA}}(n + p) \quad (301)$$

$$\frac{1}{\tau_p^{\text{TAA}}} \approx C_n^{\text{TAA}}(n + p) \quad (302)$$

A reasonable order of magnitude for the TAA coefficients C_n^{TAA} and C_p^{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 160 on page 1131](#)):

```
Physics {
    Recombination(TrapAssistedAuger ...)
    ...
}
```

The trap-assisted Auger parameters C_n^{TAA} and C_p^{TAA} are accessible in the parameter set `TrapAssistedAuger`.

Surface SRH Recombination

The surface SRH recombination model can be activated at the interface between two different materials or two different regions (see [Physics at Interfaces on page 56](#)).

At interfaces, an additional formula is used that is structurally equivalent to the bulk expression of the SRH generation–recombination:

$$R_{\text{surf, net}}^{\text{SRH}} = \frac{np - n_{i,\text{eff}}^2}{(n + n_1)/s_p + (p + p_1)/s_n} \quad (303)$$

with:

$$n_1 = n_{i,\text{eff}} \exp\left(\frac{E_{\text{trap}}}{kT}\right) \text{ and } p_1 = n_{i,\text{eff}} \exp\left(-\frac{E_{\text{trap}}}{kT}\right) \quad (304)$$

For Fermi statistics and quantization, the equations are modified in the same manner as for bulk SRH recombination (see [Eq. 279](#)).

The recombination velocities of otherwise identically prepared surfaces depend, in general, on the concentration of dopants at the surface [8][9][10]. Particularly, in cases where the doping concentration varies along an interface, it is desirable to include such a doping dependence. Sentaurus Device models doping dependence of surface recombination velocities according to:

$$s = s_0 \left[1 + s_{\text{ref}} \left(\frac{N_i}{N_{\text{ref}}} \right)^\gamma \right] \quad (305)$$

The results of Cuevas [10] indicate that for phosphorus-diffused silicon, $\gamma = 1$; while the results of King and Swanson [9] seem to 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_{trap} , s_{ref} , N_{ref} , and γ are accessible in the parameter set `SurfaceRecombination`. The corresponding values for silicon are given in [Table 59](#).

Table 59 Surface SRH parameters

Symbol	Parameter name	Electrons	Holes	Unit
s_0	S0	1×10^3	1×10^3	cm/s
s_{ref}	Sref	1×10^{-3}		1
N_{ref}	Nref	1×10^{16}		cm^{-3}

Table 59 Surface SRH parameters

Symbol	Parameter name	Electrons	Holes	Unit
γ	gamma	1		1
E_{trap}	Etrap	0		eV

The doping dependence of the recombination velocity can be suppressed by setting s_{ref} to zero.

Coupled Defect Level (CDL) Recombination

The steady state recombination rate for two coupled defect levels generalizes the familiar single-level SRH formula. An important feature of the model is a possibly increased field effect that may lead to large excess currents. The model is discussed in the literature [11].

Using CDL

The CDL recombination can be switched on using the keyword `CDL` in the `Physics` section:

```
Physics{ Recombination( CDL ... ) ... }
```

The contributions R_1 and R_2 in Eq. 308 can be plotted by 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.

CDL Model

The notation of the model parameters is illustrated in Figure 27.

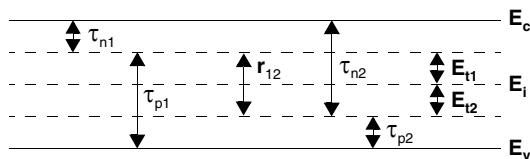


Figure 27 Notation for CDL recombination including all capture and emission processes

The CDL recombination rate is given by:

$$R = R_1 + R_2 + \left(\sqrt{R_{12}^2 - S_{12}} - R_{12} \right) \times \frac{\tau_{n1}\tau_{p2}(n+n_2)(p+p_1) - \tau_{n2}\tau_{p1}(n+n_1)(p+p_2)}{r_1r_2} \quad (306)$$

with:

$$r_j = \tau_{nj}(p + p_j) + \tau_{pj}(n + n_j) \quad (307)$$

$$R_j = \frac{np - n_{i,\text{eff}}^2}{r_j}, \quad j = 1, 2 \quad (308)$$

$$R_{12} = \frac{r_1 r_2}{2r_{12}\tau_{n1}\tau_{n2}\tau_{p1}\tau_{p2}(1-\varepsilon)} + \frac{\tau_{n1}(p+p_1) + \tau_{p2}(n+n_2)}{2\tau_{n1}\tau_{p2}(1-\varepsilon)} + \frac{\varepsilon[\tau_{n2}(p+p_2) + \tau_{p1}(n+n_1)]}{2\tau_{n2}\tau_{p1}(1-\varepsilon)} \quad (309)$$

$$S_{12} = \frac{1}{\tau_{n1}\tau_{p2}(1-\varepsilon)} \left(1 - \frac{\tau_{n1}\tau_{p2}}{\tau_{n2}\tau_{p1}} \varepsilon \right) (np - n_{i,\text{eff}}^2) \quad (310)$$

$$\varepsilon = \exp\left(-\frac{|E_{t2} - E_{t1}|}{kT}\right) \quad (311)$$

where τ_{ni} and τ_{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). The carrier lifetimes are calculated analogously to the carrier lifetimes in the SRH model (see [Shockley–Read–Hall Recombination on page 309](#)). The number of parameters is doubled compared to the SRH model, and they are changeable in the parameter set CDL.

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 [Eq. 276, p. 309](#) and [Eq. 277, p. 309](#).

For Fermi statistics and quantization, the equations are modified in the same manner as for SRH recombination (see [Eq. 279, p. 310](#)).

Radiative Recombination

Using Radiative Recombination

The radiative recombination model is activated in the `Physics` section by the keyword `Radiative`:

```
Physics {
    Recombination (Radiative)
}
```

9: Generation–Recombination

Auger Recombination

It can also be switched on or off by 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
}
```

Radiative Model

The radiative (direct) recombination model expresses the recombination rate as:

$$R_{\text{net}} = C \cdot (np - n_{i,\text{eff}}^2) \quad (312)$$

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 manner than for SRH recombination (see [Eq. 279, p. 310](#)).

Auger Recombination

The rate of band-to-band Auger recombination R_{net}^A is given by:

$$R_{\text{net}}^A = (C_n n + C_p p)(np - n_{i,\text{eff}}^2) \quad (313)$$

with temperature-dependent Auger coefficients [\[12\]\[13\]\[14\]](#):

$$C_n(T) = \left(A_{A,n} + B_{A,n} \left(\frac{T}{T_0} \right) + C_{A,n} \left(\frac{T}{T_0} \right)^2 \right) \left[1 + H_n \exp \left(-\frac{n}{N_{0,n}} \right) \right] \quad (314)$$

$$C_p(T) = \left(A_{A,p} + B_{A,p} \left(\frac{T}{T_0} \right) + C_{A,p} \left(\frac{T}{T_0} \right)^2 \right) \left[1 + H_p \exp\left(-\frac{p}{N_{0,p}}\right) \right] \quad (315)$$

where $T_0 = 300$ K. There is experimental evidence for a decrease of the Auger coefficients at high injection levels [14]. 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 Eq. 314 and Eq. 315.

Auger recombination is typically important at high carrier densities. Therefore, this injection dependence will only be seen in devices where extrinsic recombination effects are extremely 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 manner as for SRH recombination (see Eq. 279, p. 310). Default values for silicon are listed in Table 60.

Table 60 Default coefficients of 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 [cm^{-3}]
Parameter name	A	B	C	H	N0
Electrons	6.7×10^{-32}	2.45×10^{-31}	-2.2×10^{-32}	3.46667	1×10^{18}
Holes	7.2×10^{-32}	4.5×10^{-33}	2.63×10^{-32}	8.25688	1×10^{18}

Auger recombination is activated with the argument `Auger` in the `Recombination` statement:

```
Physics{ Recombination( Auger ... ) ... }
```

By default, Sentaurus Device uses Eq. 313 only if R_{net}^A is positive and replaces the value by zero if R_{net}^A is negative. To use Eq. 313 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 parameter set `Auger`.

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 α . With these coefficients for electrons and holes, the generation rate can be expressed as:

$$G^{\text{ii}} = \alpha_n n v_n + \alpha_p p v_p \quad (316)$$

Sentaurus Device implements five models of the threshold behavior of the ionization coefficients: van Overstraeten–de Man, Okuto–Crowell, Lackner, University of Bologna, and the new University of Bologna.

Sentaurus Device allows you to select the appropriate driving force for the simulation, that is, the method used to compute the accelerating field.

Using Avalanche Generation

Avalanche generation is switched on by using the keyword `Avalanche` in the `Recombination` statement in the `Physics` section. The models are selected by using the keywords `vanOverstraeten`, `Okuto`, `Lackner`, `UniBo`, and `UniBo2`. The default model is `vanOverstraeten`. For example:

```
Physics{
    Recombination(eAvalanche(CarrierTempDrive) hAvalanche(Okuto) ...
}
```

selects a driving force derived from electron temperature for electron impact ionization process and the default driving force based on `GradQuasiFermi` with the Okuto–Crowell model for holes.

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 Eq. 316 separately, specify `eAvalanche` or `hAvalanche`. To plot α_n or α_p , specify `eAlphaAvalanche` or `hAlphaAvalanche`, respectively.

van Overstraeten – de Man Model

This model is based on the Chynoweth law [15]:

$$\alpha(F_{\text{ava}}) = \gamma a \exp\left(-\frac{\gamma b}{F_{\text{ava}}}\right) \quad (317)$$

with:

$$\gamma = \frac{\tanh\left(\frac{\hbar\omega_{\text{op}}}{2kT_0}\right)}{\tanh\left(\frac{\hbar\omega_{\text{op}}}{2kT}\right)} \quad (318)$$

The factor γ 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 [16], are applicable over the range of fields $1.75 \times 10^5 \text{ Vcm}^{-1}$ to $6 \times 10^5 \text{ Vcm}^{-1}$ and are listed in [Table 61](#).

Two sets of coefficients a and b are used for high and low ranges of electric field. The values $a(\text{low})$, $b(\text{low})$ apply in the low field range $1.75 \times 10^5 \text{ Vcm}^{-1}$ to E_0 and the values $a(\text{high})$, $b(\text{high})$ apply in the high field range E_0 to $6 \times 10^5 \text{ Vcm}^{-1}$. For electrons, the impact ionization coefficients are by default the same in both field ranges.

You can adjust the coefficient values in the Sentaurus Device parameter set `vanOverstraetenDeMan`.

Table 61 Coefficients for van Overstraeten–de Man model ([Eq. 317](#))

Symbol	Parameter name	Electrons	Holes	Valid range of electric field	Unit
a	$a(\text{low})$	7.03×10^5	1.582×10^6	$1.75 \times 10^5 \text{ Vcm}^{-1}$ to E_0	cm^{-1}
	$a(\text{high})$	7.03×10^5	6.71×10^5	E_0 to $6 \times 10^5 \text{ Vcm}^{-1}$	
b	$b(\text{low})$	1.231×10^6	2.036×10^6	$1.75 \times 10^5 \text{ Vcm}^{-1}$ to E_0	V/cm
	$b(\text{high})$	1.231×10^6	1.693×10^6	E_0 to $6 \times 10^5 \text{ Vcm}^{-1}$	
E_0	$E0$	4×10^5	4×10^5		V/cm
$\hbar\omega_{\text{op}}$	hbarOmega	0.063	0.063		eV

9: Generation–Recombination

Avalanche Generation

Okuto–Crowell Model

Okuto and Crowell [17] suggested the empirical model:

$$\alpha(F_{\text{ava}}) = a \cdot (1 + c(T - T_0)) F_{\text{ava}}^\gamma \exp\left[-\left(\frac{b[1 + d(T - T_0)]}{F_{\text{ava}}}\right)^\delta\right] \quad (319)$$

where $T_0 = 300\text{ K}$ and the user-adjustable coefficients are listed in [Table 62](#) with their default values for silicon. These values are applicable to the range of electric field 10^5 Vcm^{-1} to 10^6 Vcm^{-1} . You can adjust the parameters in the parameter set Okuto.

Table 62 Coefficients for Okuto–Crowell model ([Eq. 319](#))

Symbol	Parameter name	Electrons	Holes	Unit
a	a	0.426	0.243	V^{-1}
b	b	4.81×10^5	6.53×10^5	V/cm
c	c	3.05×10^{-4}	5.35×10^{-4}	K^{-1}
d	d	6.86×10^{-4}	5.67×10^{-4}	K^{-1}
γ	gamma	1	1	1
δ	delta	2	2	1

Lackner Model

Lackner [18] derived a pseudo-local ionization rate in the form of a modification to the Chynoweth law, assuming stationary conditions. The temperature-dependent factor γ was introduced to the original model:

$$\alpha_v(F_{\text{ava}}) = \frac{\gamma a_v}{Z} \exp\left(-\frac{\gamma b_v}{F_{\text{ava}}}\right) \text{ where } v = n, p \quad (320)$$

with:

$$Z = 1 + \frac{\gamma b_n}{F_{\text{ava}}} \exp\left(-\frac{\gamma b_n}{F_{\text{ava}}}\right) + \frac{\gamma b_p}{F_{\text{ava}}} \exp\left(-\frac{\gamma b_p}{F_{\text{ava}}}\right) \quad (321)$$

and:

$$\gamma = \frac{\tanh\left(\frac{\hbar\omega_{\text{op}}}{2kT_0}\right)}{\tanh\left(\frac{\hbar\omega_{\text{op}}}{2kT}\right)} \quad (322)$$

The default values of the coefficients a , b , and $\hbar\omega_{\text{op}}$ are applicable in silicon for the range of the electric field from 10^5 Vcm^{-1} to 10^6 Vcm^{-1} .

You can adjust the coefficients in the parameter set Lackner.

Table 63 Coefficients for Lackner model ([Eq. 320](#))

Symbol	Parameter name	Electrons	Holes	Unit
a	a	1.316×10^6	1.818×10^6	cm^{-1}
b	b	1.474×10^6	2.036×10^6	V/cm
$\hbar\omega_{\text{op}}$	hbarOmega	0.063	0.063	eV

University of Bologna Impact Ionization Model

The University of Bologna impact ionization model was developed for an extended temperature range between 25°C and 400°C (see also [New University of Bologna Impact Ionization Model on page 329](#) for an updated version of this model). It is based on impact ionization data generated by the Boltzmann solver HARM [19]. It covers a wide range of electric fields (50 kVcm^{-1} to 600 kVcm^{-1}) and temperatures (300 K to 700 K). It is calibrated against impact ionization measurements [20][21] in the whole temperature range.

The model reads:

$$\alpha(F_{\text{ava}}, T) = \frac{F_{\text{ava}}}{a(T) + b(T)\exp\left[\frac{d(T)}{F_{\text{ava}} + c(T)}\right]} \quad (323)$$

The temperature dependence of the model parameters, determined by fitting experimental data, reads (for electrons):

$$a(T) = a_0 + a_1 t^{a_2} \quad b(T) = b_0 \quad c(T) = c_0 + c_1 t + c_2 t^2 \quad d(T) = d_0 + d_1 t + d_2 t^2 \quad (324)$$

9: Generation–Recombination

Avalanche Generation

and for holes:

$$a(T) = a_0 + a_1 t \quad b(T) = b_0 \exp[b_1 t] \quad c(T) = c_0 t^{c_1} \quad d(T) = d_0 + d_1 t + d_2 t^2 \quad (325)$$

where $t = T/1\text{ K}$.

[Table 64](#) lists the model parameters. The model parameters are accessible in the parameter set UniBo.

Table 64 Coefficients for University of Bologna impact ionization model

Symbol	Parameter name	Electrons	Holes	Unit
a_0	ha0	4.3383	2.376	V
a_1	ha1	-2.42×10^{-12}	1.033×10^{-2}	V
a_2	ha2	4.1233	0	1
b_0	hb0	0.235	0.17714	V
b_1	hb1	0	-2.178×10^{-3}	1
c_0	hc0	1.6831×10^4	9.47×10^{-3}	Vcm^{-1}
c_1	hc1	4.3796	2.4924	$\text{Vcm}^{-1}, 1$
c_2	hc2	0.13005	0	$\text{Vcm}^{-1}, 1$
d_0	hd0	1.2337×10^6	1.4043×10^6	Vcm^{-1}
d_1	hd1	1.2039×10^3	2.9744×10^3	Vcm^{-1}
d_2	hd2	0.56703	1.4829	Vcm^{-1}

NOTE When the University of Bologna impact ionization model is selected for both carriers, that is, Recombination(Avalanche(UniBo)), Sentaurus Device also enables Auger generation (see [Auger Recombination on page 322](#)). Specify Auger(-WithGeneration) to disable this generation term if necessary.

New University of Bologna Impact Ionization Model

The impact ionization model described in [University of Bologna Impact Ionization Model on page 327](#) was developed further in [\[22\]](#)[\[23\]](#)[\[24\]](#) 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 [\[19\]](#) and is calibrated against specially designed impact ionization measurements [\[20\]](#)[\[21\]](#). It covers a wide range of electric fields. The model reads:

$$\alpha(F_{\text{ava}}, T) = \frac{F_{\text{ava}}}{a(T) + b(T)\exp\left[\frac{d(T)}{F_{\text{ava}} + c(T)}\right]} \quad (326)$$

where the coefficients a , b , c , and d are polynomials of T :

$$a(T) = \sum_{k=0}^5 a_k \left(\frac{T}{1\text{K}}\right)^k \quad (327)$$

$$b(T) = \sum_{k=0}^{10} b_k \left(\frac{T}{1\text{K}}\right)^k \quad (328)$$

$$c(T) = \sum_{k=0}^5 c_k \left(\frac{T}{1\text{K}}\right)^k \quad (329)$$

$$d(T) = \sum_{k=0}^5 d_k \left(\frac{T}{1\text{K}}\right)^k \quad (330)$$

[Table 65](#) lists the model parameters.

Table 65 Coefficients for UniBo2 impact ionization model

Symbol	Electrons		Holes		Unit
	Parameter Name	Value	Parameter Name	Value	
a_0	$a0_e$	4.65403	$a0_h$	2.26018	V
a_1	$a1_e$	-8.76031×10^{-3}	$a1_h$	0.0134001	V
a_2	$a2_e$	1.34037×10^{-5}	$a2_h$	-5.87724×10^{-6}	V
a_3	$a3_e$	-2.75108×10^{-9}	$a3_h$	-1.14021×10^{-9}	V
b_0	$b0_e$	-0.128302	$b0_h$	0.058547	V
b_1	$b1_e$	4.45552×10^{-3}	$b1_h$	-1.95755×10^{-4}	V

9: Generation–Recombination

Avalanche Generation

Table 65 Coefficients for UniBo2 impact ionization model

Symbol	Electrons		Holes		Unit
	Parameter Name	Value	Parameter Name	Value	
b_2	$b2_e$	-1.0866×10^{-5}	$b2_h$	2.44357×10^{-7}	V
b_3	$b3_e$	9.23119×10^{-9}	$b3_h$	-1.33202×10^{-10}	V
b_4	$b4_e$	-1.82482×10^{-12}	$b4_h$	2.68082×10^{-14}	V
b_5	$b5_e$	-4.82689×10^{-15}	$b5_h$	0	V
b_6	$b6_e$	1.09402×10^{-17}	$b6_h$	0	V
b_7	$b7_e$	-1.24961×10^{-20}	$b7_h$	0	V
b_8	$b8_e$	7.55584×10^{-24}	$b8_h$	0	V
b_9	$b9_e$	-2.28615×10^{-27}	$b9_h$	0	V
b_{10}	$b10_e$	2.73344×10^{-31}	$b10_h$	0	V
c_0	$c0_e$	7.76221×10^3	$c0_h$	1.95399×10^4	Vcm^{-1}
c_1	$c1_e$	25.18888	$c1_h$	-104.441	Vcm^{-1}
c_2	$c2_e$	-1.37417×10^{-3}	$c2_h$	0.498768	Vcm^{-1}
c_3	$c3_e$	1.59525×10^{-4}	$c3_h$	0	Vcm^{-1}
d_0	$d0_e$	7.10481×10^5	$d0_h$	2.07712×10^6	Vcm^{-1}
d_1	$d1_e$	3.98594×10^3	$d1_h$	993.153	Vcm^{-1}
d_3	$d2_e$	-7.19956	$d2_h$	7.77769	Vcm^{-1}
d_3	$d3_e$	6.96431×10^{-3}	$d3_h$	0	Vcm^{-1}

The model parameters can be specified in the Sentaurus Device parameter file:

```
UniBo2 {
    a0_e = 4.65403
    a0_h = 2.26018
    ...
}
```

NOTE The name of the UniBo2 model is case sensitive. Both in the command file and parameter file of Sentaurus Device, the exact spelling UniBo2 must be used.

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, $F_{ava} = \vec{F} \cdot \hat{J}_{n,p}$, (keyword Eparallel), or the value of the gradient of the quasi-Fermi level, $F_{ava} = |\nabla \Phi_{n,p}|$, (keyword GradQuasiFermi). For hydrodynamic simulations, F_{ava} can be computed from the carrier temperature (keyword CarrierTempDrive, see [Avalanche Generation with Hydrodynamic Transport on page 331](#)). The default is GradQuasiFermi. See [Table 162 on page 1133](#) for a summary of keywords.

The option ElectricField is used to perform breakdown simulations using the ‘ionization integral’ method (see [Approximate Breakdown Analysis: Poisson Equation Approach on page 333](#)).

Avalanche Generation with Hydrodynamic Transport

If the hydrodynamic transport model is used, it is also possible (and usually recommended) to select a local carrier temperature-dependent impact ionization model, where the driving force F_{ava} equals an effective field E^{eff} obtained from the carrier temperature. This is achieved by using the construct Avalanche(CarrierTempDrive) as an option to Recombination(Avalanche) in the Physics section:

```
Physics{ Hydro
          Recombination(Avalanche(CarrierTempDrive)) ... }
```

Otherwise, the driving force computation is still based on the GradQuasiFermi method.

The usual conversion of local carrier temperatures to effective fields E^{eff} is described by the algebraic equations:

$$n\mu_n(E_n^{\text{eff}})^2 = n \frac{3kT_n - T}{2q\lambda_n\tau_{en}} \quad (331)$$

$$p\mu_p(E_p^{\text{eff}})^2 = p \frac{3kT_p - T}{2q\lambda_p\tau_{ep}} \quad (332)$$

which are obtained from the energy conservation equation under time-independent, homogeneous conditions. [Eq. 331](#) and [Eq. 332](#) 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.

9: Generation–Recombination

Avalanche Generation

The parameters λ_n and λ_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 [Eq. 331](#) and [Eq. 332](#) can be activated by specifying parameters $\Upsilon_n = 0$, $\Upsilon_p = 0$ in the same `AvalancheFactors` section.

The simplified conversion formulas discussed above 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^{eff} .

Such heat sinks arise from nonelastic processes, such as the impact ionization itself. Sentaurus Device supports the following model to account for these heat sinks:

$$n v_{\text{sat},n} E_n^{\text{eff}} = n \frac{3kT_n - T}{2q\lambda_n\tau_{en}} + \frac{\Upsilon_n}{q}(E_g + \delta_n kT_n)\alpha_n n v_{\text{sat},n} \quad (333)$$

A similar equation E_p^{eff} is used to determine E_p^{eff} . To activate this model, set the parameters Υ_n and Υ_p to 1. This is the default for silicon, where the generalized conversion formula [Eq. 333](#) gives good agreement with Monte Carlo data for $\delta_n = \delta_p = 3/2$. For all other materials, the default of the parameters Υ_n and Υ_p is 0.

Table 66 Hydrodynamic avalanche model: Default parameters

Symbol	Parameter name	Default value
λ_n	<code>n_1_f</code>	1
λ_p	<code>p_1_f</code>	1
Υ_n	<code>n_gamma</code>	1
Υ_p	<code>p_gamma</code>	1
δ_n	<code>n_delta</code>	1.5
δ_p	<code>p_delta</code>	1.5

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.

Approximate Breakdown Analysis: Poisson Equation Approach

Junction breakdown due to avalanche generation is simulated by inspecting the ionization integrals:

$$I_n = \int_0^W \alpha_n(x) e^{-\int_x^W (\alpha_n(x') - \alpha_p(x')) dx'} dx \quad (334)$$

$$I_p = \int_0^W \alpha_p(x) e^{-\int_0^x (\alpha_p(x') - \alpha_n(x')) dx'} dx \quad (335)$$

where α_n , α_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 an ionization integral equals one. Eq. 334 describes electron injection (electron primary current) and Eq. 335 describes hole injection. Since these breakdown criteria do not depend on current densities, a breakdown analysis can be performed 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. This driving force is less physical than the others available in Sentaurus Device. Therefore, Synopsys discourages using it for any purpose other than approximate breakdown analysis.

`ComputeIonizationIntegrals` in the `Math` section switches on the computation of the ionization integrals for ionization paths crossing the local field maxima in the semiconductor. By default, Sentaurus Device reports only the path with the largest I_{mean} . With the addition of the keyword `WriteAll`, information about the ionization integrals for all computed paths is written to the log file.

The `Math` keyword `BreakAtIonIntegral` is used to terminate the quasistationary simulation when the largest ionization integral is greater than one.

9: Generation–Recombination

Approximate Breakdown Analysis: Poisson Equation Approach

The complete syntax of this keyword is `BreakAtIonIntegral(<number> <value>)` where a quasistationary simulation finishes if the number ionization integral is greater than value, and the ionization integrals are ordered with respect to decreasing value:

```
Math { BreakAtIonIntegral }
```

Three optional keywords in the `Plot` section specify the values of the corresponding ionization integrals that are stored along the breakdown paths:

```
Plot { eIonIntegral | hIonIntegral | MeanIonIntegral }
```

These ion integrals can be visualized by using Tecplot SV. A typical command file of Sentaurus Device is:

```
Electrode {
    { name="anode" Voltage=0 }
    { name="cathode" Voltage=600 }
}
File {
    grid      = "@grid@"
    doping    = "@doping@"
    current   = "@plot@"
    output    = "@log@"
    plot      = "@data@"
}
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(WriteAll)
}
Plot {
    eIonIntegral hIonIntegral MeanIonIntegral
    eDensity hDensity
    ElectricField/Vector
    eAlphaAvalanche hAlphaAvalanche
}
```

Band-to-Band Tunneling Models

Sentaurus Device provides several band-to-band tunneling models:

- Schenk model (see [Schenk Model on page 336](#)).
- Hurkx model (see [Hurkx Band-to-Band Model on page 338](#)).
- A family of simple models (see [Simple Band-to-Band Models on page 337](#)).
- Nonlocal path model (see [Dynamic Nonlocal Path Band-to-Band Model on page 339](#)).

The Schenk, 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 344](#)).

Using Band-to-Band Tunneling

Band-to-band tunneling is controlled by the Band2Band option of Recombination:

```
Recombination( ...
    Band2Band (
        Model = Schenk | Hurkx | E1 | E1_5 | E2 | NonlocalPath1 | NonlocalPath2 |
        NonlocalPath3
        DensityCorrection = Local | None
        InterfaceReflection | -InterfaceReflection
        FranzDispersion | -FranzDispersion
    )
)
```

Model determines the model:

- Schenk selects the Schenk model.
- Hurkx selects the Hurkx model.
- E1, E1_5, and E2 select one of the simple models.
- NonlocalPath1, NonlocalPath2, and NonlocalPath3 select the nonlocal path model with the number of tunneling processes equal to 1, 2, and 3, respectively.

DensityCorrection is used by the Schenk, 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 337](#)).

InterfaceReflection is used by the nonlocal path models, and this option 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.

FranzDispersion is used by the nonlocal path models, and this option is switched off by default. When it is switched on, the Franz dispersion relation (Eq. 519, p. 524) instead of the Kane dispersion relation (Eq. 344, p. 340) for the imaginary wave vector is used in the direct tunneling process. The indirect tunneling process is not affected by this option.

For backward compatibility:

- Band2Band alone (without 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 models are available in the Band2BandTunneling parameter set. The parameters specific to individual models are described in the respective sections. The parameter MinField (specified in Vcm^{-1}) is used by the Schenk, Hurkx, and simple models for smoothing at small electric fields. A value of zero (the default) disables smoothing.

Schenk 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 (see SRH Field Enhancement on page 313) must also be switched on.

Band-to-band tunneling is modeled using the expression [25]:

$$R_{\text{net}}^{\text{bb}} = AF^{7/2} \frac{\tilde{n}\tilde{p} - n_{i,\text{eff}}^2}{(\tilde{n} + n_{i,\text{eff}})(\tilde{p} + n_{i,\text{eff}})} \left[\frac{(F_C^\mp)^{-3/2} \exp\left(-\frac{F_C^\mp}{F}\right)}{\exp\left(\frac{\hbar\omega}{kT}\right) - 1} + \frac{(F_C^\pm)^{-3/2} \exp\left(-\frac{F_C^\pm}{F}\right)}{1 - \exp\left(-\frac{\hbar\omega}{kT}\right)} \right] \quad (336)$$

where \tilde{n} and \tilde{p} equal n and p for DensityCorrection=None, and are given by Eq. 338 for DensityCorrection=Local. The critical field strengths read:

$$F_C^\pm = B(E_{g,\text{eff}} \pm \hbar\omega)^{3/2} \quad (337)$$

The upper sign in Eq. 336 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, Eq. 336 is modified in the same way as for SRH recombination (see Eq. 279, p. 310).

The parameters [25] are given in Table 67 and can be accessed in the parameter set Band2BandTunneling. The defaults were obtained assuming the field direction to be $\langle 111 \rangle$.

Table 67 Coefficients for band-to-band tunneling (Schenk model)

Symbol	Parameter name	Default value	Unit
A	A	8.977×10^{20}	$\text{cm}^{-1}\text{s}^{-1}\text{V}^{-2}$
B	B	2.14667×10^7	$\text{V}\text{cm}^{-1}\text{eV}^{-3/2}$
$\hbar\omega$	hbarOmega	18.6	meV

Schenk Density Correction

The modified electron density reads:

$$\tilde{n} = n \left(\frac{n_{i,\text{eff}}}{N_C} \right)^{\frac{\gamma_n |\nabla E_{F,n}|}{F}} \quad (338)$$

and 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 Schenk TAT Density Correction on page 316. The reference densities n_{ref} and p_{ref} are specified (in cm^{-3}) by the DenCorRef parameter pair in the Band2BandTunneling parameter set. An additional parameter MinGradQF (specified in eVcm^{-1}) is available for smoothing at small values of the gradient for the Fermi potential.

Simple Band-to-Band Models

Sentaurus Device provides a family of simple band-to-band models. Compared to advanced models, the most striking 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 [26] as:

$$G^{\text{b2b}} = AF^P \exp\left(-\frac{B}{F}\right) \quad (339)$$

Depending on value of Model, P takes the values 1, 1.5, or 2.

[Table 68](#) lists the coefficients of models and their defaults. The coefficients A and B can be changed in the parameter set Band2BandTunneling.

Table 68 Coefficients for band-to-band tunneling (simple 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	$3.4 \times 10^{21} \text{ cm}^{-1} \text{ s}^{-1} \text{ V}^{-2}$	$22.6 \times 10^6 \text{ Vcm}^{-1}$

Hurkx Band-to-Band Model

Similar to the other band-to-band tunneling models, in the Hurkx model [27], the tunneling carriers are modeled by an additional generation–recombination process. Its contribution is expressed as:

$$R_{\text{net}}^{\text{bb}} = A \cdot D \cdot \left(\frac{F}{1 \text{ V/cm}} \right)^P \exp \left(-\frac{BE_g(T)^{3/2}}{E_g(300\text{K})^{3/2} F} \right) \quad (340)$$

where:

$$D = \frac{np - n_{i,\text{eff}}^2}{(n + n_{i,\text{eff}})(p + n_{i,\text{eff}})} (1 - |\alpha|) + \alpha \quad (341)$$

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, [Eq. 341](#) is modified in the same way as for SRH recombination (see [Eq. 279](#)).

For DensityCorrection=Local, n and p in [Eq. 341](#) are replaced by \tilde{n} and \tilde{p} (see [Schenk Density Correction on page 337](#)).

The coefficients A (in $\text{cm}^{-3}\text{s}^{-1}$), B (in V/cm), P , and α can be specified in the Band2BandTunneling parameter set. By default, Sentaurus Device uses $\alpha = 0$ and the parameters from the E2 model (see [Table 68 on page 338](#)). Different values for the generation (Agen, Bgen, Pgen) and recombination (Arec, Brec, Prec) of carriers are supported. For example, to change the parameters to those used in [27], 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]
}
```

Dynamic Nonlocal Path Band-to-Band Model

The present model implements the nonlocal generation of electrons and holes caused by direct and phonon-assisted band-to-band tunneling processes [28]. 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 the present model. The model can be applied to heterostructure devices with abrupt and graded heterojunctions.

The main difference between the present model and the band-to-band tunneling model based on the nonlocal mesh (see [Band-to-Band Contributions to Nonlocal Tunneling Current on page 527](#)) is that the tunneling path is determined dynamically based on the energy band profile rather than predefined by the nonlocal mesh. Therefore, the present model does not require the user specification of the nonlocal mesh.

The model dynamically searches for the tunneling path with the following assumptions:

- The tunneling path is a straight line with its direction opposite to the gradient of the valence band at the starting position.
- The tunneling energy is equal to the valence band energy at the starting position and is equal to the conduction band energy plus band offset at the ending position.
- When the tunneling path encounters Neumann boundaries or semiconductor–insulator interfaces, it undergoes specular reflection.

NOTE The present model is not suitable for AC or noise analysis as nonlocal derivative terms in the Jacobian matrix are not taken into account.

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:

$$R_{\text{net}}^{\text{d}} = |\nabla E_V(0)| C_{\text{d}} \exp\left(-2 \int_0^l \kappa dx\right) \left[\left(\exp\left[\frac{\varepsilon - E_{F,n}(l)}{kT(l)}\right] + 1 \right)^{-1} - \left(\exp\left[\frac{\varepsilon - E_{F,p}(0)}{kT(0)}\right] + 1 \right)^{-1} \right] \quad (342)$$

$$C_d = \frac{g\pi}{36h} \left(\int_0^l \frac{dx}{\kappa} \right)^{-1} \left[1 - \exp \left(-k_m^2 \int_0^l \frac{dx}{\kappa} \right) \right] \quad (343)$$

where h is Planck's constant, g is the degeneracy factor, $\varepsilon = E_V(0) = E_C(l) + \Delta_C(l)$ is the tunneling energy, Δ_C is the conduction band offset, and κ is the magnitude of the imaginary wave vector obtained from the Kane two-band dispersion relation [28]:

$$\kappa = \frac{1}{\hbar} \sqrt{m_r E_{g,tun}(1 - \alpha^2)} \quad (344)$$

$$\alpha = -\frac{m_0}{2m_r} + 2 \sqrt{\frac{m_0}{2m_r} \left(\frac{\varepsilon - E_V}{E_{g,tun}} - \frac{1}{2} \right) + \frac{m_0^2}{16m_r^2} + \frac{1}{4}} \quad (345)$$

$$\frac{1}{m_r} = \frac{1}{m_V} + \frac{1}{m_C} \quad (346)$$

$E_{g,tun} = E_{g,eff} + \Delta_C$ is the effective band gap including the band offset, and k_m is the maximum transverse momentum determined by the maximum valence-band energy ε_{max} and the minimum conduction-band energy ε_{min} :

$$k_m^2 = \min(k_{vm}^2, k_{cm}^2) \quad (347)$$

$$k_{vm}^2 = \frac{2m_V(\varepsilon_{max} - \varepsilon)}{\hbar^2} \quad (348)$$

$$k_{cm}^2 = \frac{2m_C(\varepsilon - \varepsilon_{min})}{\hbar^2} \quad (349)$$

In the Kane two-band dispersion relation, the effective mass in the conduction band m_C and the valence band m_V are related [28]:

$$\frac{1}{m_C} = \frac{1}{2m_r} + \frac{1}{m_0} \quad (350)$$

$$\frac{1}{m_V} = \frac{1}{2m_r} - \frac{1}{m_0} \quad (351)$$

The net hole recombination rate due to the phonon-assisted band-to-band tunneling process $R_{\text{net}}^{\text{p}}$ can be written as:

$$R_{\text{net}}^{\text{p}} = |\nabla E_V(0)| C_p \exp \left(-2 \int_0^{x_0} \kappa_V dx - 2 \int_{x_0}^l \kappa_C dx \right) \left[\left(\exp \left[\frac{\varepsilon - E_{F,n}(l)}{kT(l)} \right] + 1 \right)^{-1} - \left(\exp \left[\frac{\varepsilon - E_{F,p}(0)}{kT(0)} \right] + 1 \right)^{-1} \right] \quad (352)$$

$$C_p = \int_0^l \frac{g(1+2N_{\text{op}})D_{\text{op}}^2}{2^6 \pi^2 \rho \varepsilon_{\text{op}} E_{g,\text{tun}}} \sqrt{\frac{m_V m_C}{h l \sqrt{2m_r} E_{g,\text{tun}}}} dx \left(\int_0^{x_0} \frac{dx}{\kappa_V} \right)^{-1} \left(\int_{x_0}^l \frac{dx}{\kappa_C} \right)^{-1} \left[1 - \exp \left(-k_{\text{vm}}^2 \int_0^{x_0} \frac{dx}{\kappa_V} \right) \right] \left[1 - \exp \left(-k_{\text{cm}}^2 \int_{x_0}^l \frac{dx}{\kappa_C} \right) \right] \quad (353)$$

where D_{op} , ε_{op} , and $N_{\text{op}} = [\exp(\varepsilon_{\text{op}}/kT) - 1]^{-1}$ are the deformation potential, energy, and number of optical phonons, respectively, ρ is the mass density, and κ_V and κ_C are the magnitude of the imaginary wave vectors from the Keldysh dispersion relation:

$$\kappa_V = \frac{1}{\hbar} \sqrt{2m_V |\varepsilon - E_V|} \Theta(\varepsilon - E_V) \quad (354)$$

$$\kappa_C = \frac{1}{\hbar} \sqrt{2m_C |E_C + \Delta_C - \varepsilon|} \Theta(E_C + \Delta_C - \varepsilon) \quad (355)$$

and x_0 is the location where $\kappa_V = \kappa_C$.

As [Eq. 342, p. 339](#) and [Eq. 352](#) are the extension of the results in [28] to arbitrary band profiles, these expressions are reduced to the well-known Kane and Keldysh models in the uniform electric-field limit [28]:

$$R_{\text{net}} = A \left(\frac{F}{F_0} \right)^P \exp \left(-\frac{B}{F} \right) \quad (356)$$

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 [28]:

$$A = \frac{g \pi m_r^{1/2} (qF_0)^2}{9h^2 [E_g(300\text{K}) + \Delta_C]^{1/2}} \quad (357)$$

$$B = \frac{\pi^2 m_r^{1/2} [E_g(300\text{K}) + \Delta_C]^{3/2}}{qh} \quad (358)$$

For the phonon-assisted tunneling process, A and B can be expressed by [28]:

$$A = \frac{g(m_v m_c)^{3/2} (1 + 2N_{op}) D_{op}^2 (qF_0)^{5/2}}{2^{21/4} h^{5/2} m_r^{5/4} \rho \varepsilon_{op} [E_g(300K) + \Delta_c]^{7/4}} \quad (359)$$

$$B = \frac{2^{7/2} \pi m_r^{1/2} [E_g(300K) + \Delta_c]^{3/2}}{3qh} \quad (360)$$

Using Nonlocal Path Band-to-Band Model

The parameters for the nonlocal path band-to-band model are available in the parameter set `Band2BandTunneling`. To make the parameter set of the model consistent with the existing band-to-band tunneling models, the prefactor A and the exponential factor B are chosen as the input parameters. In addition, the conduction band offset Δ_c , phonon energy ε_{op} , and the effective mass ratio m_v/m_c can be specified.

Specifying $\varepsilon_{op} = 0$ selects the direct tunneling process, and parameters A , B , and Δ_c determine g and m_r from [Eq. 357, p. 341](#) and [Eq. 358, p. 341](#).

When $\varepsilon_{op} = 0$ and the option `FranzDispersion` is switched on in the `Band2Band` option of the command file, the magnitude of the imaginary wave vector is obtained from the Franz two-band dispersion relation ([Eq. 519, p. 524](#)) instead of the Kane two-band dispersion relation ([Eq. 344, p. 340](#)). In this case, g and m_r are still obtained from [Eq. 357](#) and [Eq. 358](#). Then, [Eq. 346, p. 340](#) and m_v/m_c determine m_v and m_c .

Specifying $\varepsilon_{op} > 0$ selects the phonon-assisted tunneling process, and parameters A , B , Δ_c , ε_{op} , and m_v/m_c determine gD_{op}^2/ρ , m_v , and m_c from [Eq. 359](#) and [Eq. 360](#). When $m_v/m_c = 0$, m_v and m_c are determined from [Eq. 350, p. 340](#) and [Eq. 351, p. 340](#).

Up to three different tunneling paths can be specified in the parameter file, and they are activated by setting `Model=NonlocalPath1 | NonlocalPath2 | NonlocalPath3` in the `Band2Band` option of the command file. `NonlocalPath1` selects the first tunneling path, `NonlocalPath2` selects the first and second tunneling paths, and `NonlocalPath3` selects all three tunneling paths.

NOTE Each tunneling path must have a consistent tunneling process (either direct or phonon-assisted tunneling process) in different regions. For example, specifying `Ppath1=0` (direct tunneling) in silicon regions and specifying `Ppath1` greater than zero (phonon-assisted tunneling) in polysilicon regions will induce 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, the band-to-band tunneling is neglected.

Table 69 lists the coefficients of models and their defaults. These parameters can be mole fraction–dependent. The default parameters A and B are obtained from [27].

Table 69 Default parameters for nonlocal path band-to-band tunneling model

Symbol	Path index	Parameter name	Default value	Unit
A	1	<code>Apath1</code>	4×10^{14}	$\text{cm}^{-3}\text{s}^{-1}$
B	1	<code>Bpath1</code>	1.9×10^7	Vcm^{-1}
Δ_C	1	<code>Dpath1</code>	0	eV
ε_{op}	1	<code>Ppath1</code>	0.037	eV
m_V/m_C	1	<code>Rpath1</code>	0	1
A	2	<code>Apath2</code>	0	$\text{cm}^{-3}\text{s}^{-1}$
B	2	<code>Bpath2</code>	0	Vcm^{-1}
Δ_C	2	<code>Dpath2</code>	0	eV
ε_{op}	2	<code>Ppath2</code>	0	eV
m_V/m_C	2	<code>Rpath2</code>	0	1
A	3	<code>Apath3</code>	0	$\text{cm}^{-3}\text{s}^{-1}$
B	3	<code>Bpath3</code>	0	Vcm^{-1}
Δ_C	3	<code>Dpath3</code>	0	eV
ε_{op}	3	<code>Ppath3</code>	0	eV
m_V/m_C	3	<code>Rpath3</code>	0	1

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`.

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, [Eq. 336](#), [Eq. 339](#), and [Eq. 340](#) 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, two additional control parameters are introduced in the `Band2BandTunneling` parameter set:

```
dDist = <value> # [cm]
dPot = <value> # [V]
```

By default, both these parameters equal zero. Sentaurus Device disables 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 disables band-to-band tunneling at each point where the change of the electrostatic potential in field direction within a distance `dPot/F` is smaller than `dPot/2`.

Bimolecular Recombination

The bimolecular recombination model describes the interaction of electron–hole pairs and singlet excitons (see [Singlet Exciton Equation on page 189](#)).

Physical Model

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:

$$R_{\text{bimolec}} = \gamma \cdot \frac{q}{\epsilon_0 \epsilon_r} \cdot (\mu_n + \mu_p) \left(np - n_{\text{eff}}^2 \frac{n_{\text{se}}}{n_{\text{eq}}} \right) \quad (361)$$

where γ is a prefactor for singlet exciton, q is the elementary charge, and ϵ_0 and ϵ_r denote the free space and relative permittivities, respectively. Electron and hole mobilities are given by μ_n and μ_p , accordingly. n , p , and n_{eff} describe the electron, hole, and effective intrinsic density,

respectively. Furthermore, n_{se} is the singlet exciton density, and $n_{\text{se}}^{\text{eq}}$ denotes the singlet exciton equilibrium density.

Using Bimolecular Recombination

The bimolecular recombination model is activated by using the keyword `Bimolecular` as an argument of the `Recombination` statement in the `SingletExciton` section (see [Table 177 on page 1146](#)). It is switched off by default and can be activated regionwise (see [Singlet Exciton Equation on page 189](#)). An example is:

```
Physics (Region="EML-ETL") {
    SingletExciton (
        Recombination ( Bimolecular )
    )
}
```

[Table 70](#) lists the parameter of the bimolecular recombination model, which is accessible in the `SingletExciton` section of the parameter file.

Table 70 Default parameter for bimolecular recombination

Symbol	Parameter name	Default value	Unit
γ	gamma	0.25	1

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] 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.

9: Generation–Recombination

References

- [6] 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.
- [7] 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.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] 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.
- [15] A. G. Chynoweth, “Ionization Rates for Electrons and Holes in Silicon,” *Physical Review*, vol. 109, no. 5, pp. 1537–1540, 1958.
- [16] 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.
- [17] 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.
- [18] T. Lackner, “Avalanche Multiplication in Semiconductors: A Modification of Chynoweth’s Law,” *Solid-State Electronics*, vol. 34, no. 1, pp. 33–42, 1991.
- [19] 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.

- [20] 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.
- [21] 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.
- [22] 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.
- [23] 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.
- [24] 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.
- [25] 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.
- [26] J. J. Liou, “Modeling the Tunnelling Current in Reverse-Biased p/n Junctions,” *Solid-State Electronics*, vol. 33, no. 7, pp. 971–972, 1990.
- [27] 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.
- [28] E. O. Kane, “Theory of Tunneling,” *Journal of Applied Physics*, vol. 32, no. 1, pp. 83–91, 1961.

9: Generation–Recombination

References

CHAPTER 10 Traps and Fixed Charges

This chapter presents information on how traps are handled by Sentaurus Device.

Traps are important in device physics. For example, they provide doping, enhance recombination, and increase leakage through insulators. Several models (for example, Shockley–Read–Hall recombination, described in [Shockley–Read–Hall Recombination on page 309](#)) 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 (insulator) fixed charges.

Sentaurus Device provides several trap types (electron and hole traps, fixed charges), five types of energetic distribution (level, uniform, exponential, Gaussian, and table), and various models for capture and emission rates, including the V-model (optionally, with field-dependent cross sections), the J-model, and nonlocal tunneling. Traps are available for both bulk and interfaces.

Basic Syntax for Traps

The specification of trap distributions and trapping models appears in the `Physics` section. In contrast to other models, most model parameters are specified here as well. 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> )
        ...
    )
}
```

and likewise for `Material`, `RegionInterface`, and `MaterialInterface`. Above, each `<trap specification>` describes one particular trap distribution, and the models and parameters applied to it. When only a single trap specification is present, the inner pair of parentheses can be omitted. [Table 224 on page 1176](#) summarizes the options that can appear in the trap specification.

10: Traps and Fixed Charges

Trap Types

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` statement.

Trap Types

The keywords `FixedCharge`, `Acceptor`, `Donor`, `eNeutral`, and `hNeutral` select the type of trap distribution:

- `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 keywords `Level`, `Uniform`, `Exponential`, `Gaussian`, and `Table` determine the energetic distribution of traps. They select a single-energy level, a uniform distribution, an exponential distribution, a Gaussian distribution, and a user-defined table distribution, respectively:

$$\begin{aligned} N_0 & \quad \text{for } E = E_0 & & \text{for Level} \\ N_0 & \quad \text{for } E_0 - 0.5E_S < E < E_0 + 0.5E_S & & \text{for Uniform} \\ N_0 \exp\left(-\frac{|E - E_0|}{E_S}\right) & & & \text{for Exponential} \\ N_0 \exp\left(-\frac{(E - E_0)^2}{2E_S^2}\right) & & & \text{for Gaussian} \\ \begin{cases} N_1 & \text{for } E = E_1 \\ \dots & \dots \\ N_m & \text{for } E = E_m \end{cases} & & & \text{for Table} \end{aligned} \tag{362}$$

N_0 is set with the `Conc` keyword. For a `Level` distribution, `Conc` is given in cm^{-3} (for regionwise or materialwise specifications) or cm^{-2} (for interface-wise specifications). For the other energetic distributions, `Conc` is given in $\text{eV}^{-1}\text{cm}^{-3}$ or $\text{eV}^{-1}\text{cm}^{-2}$. For `FixedCharge`, the sign of `Conc` denotes the sign of the fixed charges. For the other trap types, `Conc` must not be negative.

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 keywords `fromCondBand`, `fromMidBandGap`, or `fromValBand`, the energy levels in the table are relative to the conduction band, intrinsic energy, or valence band, respectively. When the Table distribution contains only one energy–concentration pair, it degenerates into a Level distribution.

E_0 and E_s are given in eV by `EnergyMid` and `EnergySig`. The energy of the center of the trap distribution, E_{trap}^0 , is obtained from E_0 depending on the presence of one of the keywords `fromCondBand`, `fromMidBandGap`, or `fromValBand`:

$$E_{\text{trap}}^0 = \begin{cases} E_C - E_0 - E_{\text{shift}} & \text{fromCondBand} \\ [E_C + E_V + kT \ln(N_V/N_C)]/2 + E_0 + E_{\text{shift}} & \text{fromMidBandGap} \\ E_V + E_0 + E_{\text{shift}} & \text{fromValBand} \end{cases} \quad (363)$$

Internally, Sentaurus Device approximates trap-energy distributions by discrete energy levels. The number of levels defaults to 15 and is set by `TrapDLN` in the Math section.

In Eq. 363, E_{shift} is zero (the default) or is computed by a PMI specified with `EnergyShift=<model name>` or `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, with `ReferencePoint=<vector>`, you can specify a coordinate in the device from where these quantities are to be taken. For more details, see [Trap Energy Shift on page 1011](#).

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 rate due to the traps will be fully accounted for on that side. Without this side specification, the energy parameters refer to the lower bandgap material; the charge and recombination rates due to traps are evenly distributed on both sides.

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. Otherwise, it is scaled by its maximum and multiplied by N_0 . Datasets available for `Sfactor` are `DeepLevels`, `xMoleFraction`, and `yMoleFraction` (read from the doping file), and `eTrappedCharge` and `hTrappedCharge` (read from the file specified by `DevFields` in the File section), as well as the PMI user fields (read from the file specified by `PMIUserFields` in the File section).

Alternatively, with `Sfactor = "<model name>"`, the spatial distribution is computed using the PMI. See [Trap Space Factor on page 1006](#) for more details. During a transient simulation,

10: Traps and Fixed Charges

Trap Models and Parameters

this PMI gives the possibility to have a time-dependent trap concentration. In this case, spatial distribution can be computed based on the solution from the previous time step available through the PMI.

SpatialShape selects a multiplicative modifier function for the trap concentration. If SpatialShape is Uniform (the default), the multiplier for point (x, y, z) is:

$$\Theta(\sigma_x - |x - x_0|) \Theta(\sigma_y - |y - y_0|) \Theta(\sigma_z - |z - z_0|) \quad (364)$$

If SpatialShape is Gaussian, the multiplier is:

$$\exp\left(-\frac{(x - x_0)^2}{2\sigma_x^2} - \frac{(y - y_0)^2}{2\sigma_y^2} - \frac{(z - z_0)^2}{2\sigma_z^2}\right) \quad (365)$$

In Eq. 364 and Eq. 365, (x_0, y_0, z_0) and $(\sigma_x, \sigma_y, \sigma_z)$ are given (in μm) by the SpaceMid and SpaceSig options, respectively. By default, the components of SpaceSig are huge, so that the multiplier becomes one.

Trap Models and Parameters

Trap Occupation Dynamics

The electron occupation f^n of an eNeutral or Acceptor trap is a number between 0 and 1, and changes due to the capture and emission of electrons:

$$\frac{\partial f^n}{\partial t} = \sum_i r_i^n \quad (366)$$

$$r_i^n = (1 - f^n)c_i^n - f^n e_i^n \quad (367)$$

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, respectively. The sum in Eq. 366 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 Eq. 366 vanishes. The occupation becomes:

$$f^n = \frac{\sum c_i^n}{\sum (c_i^n + e_i^n)} \quad (368)$$

and the net electron capture rate due to process k becomes:

$$r_k^n = \frac{c_k^n \sum e_i^n - e_k^n \sum c_i^n}{\sum (c_i^n + e_i^n)} \quad (369)$$

Analogous equations hold for hNeutral and Donor traps (use $c_i^p = e_i^n$ and $e_i^p = c_i^n$ to derive them).

In particular, in the stationary state, for traps with concentration N_0 , the V-model (see [Local Trap Capture and Emission on page 354](#)) and [Eq. 369](#) lead to the Shockley–Read–Hall recombination rate:

$$R_{\text{net}} = \frac{N_0 v_{\text{th}}^n v_{\text{th}}^p \sigma_n \sigma_p (np - n_{i,\text{eff}}^2)}{v_{\text{th}}^n \sigma_n (n + n_1/g_n) + v_{\text{th}}^p \sigma_p (p + p_1/g_p)} \quad (370)$$

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:

$$e_i = \frac{c_i}{g} \exp\left(\frac{E_{\text{trap}} - E_F^i}{kT_i}\right) \quad (371)$$

Here, E_{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, and 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 Trap Capture and Emission](#)) or nonlocal (see [Tunneling and Traps on page 357](#)). 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.

10: Traps and Fixed Charges

Trap Models and Parameters

Local Trap Capture and Emission

The electron capture rate from the conduction band at the same location as the trap is:

$$c_C^n = \sigma_n \left[(1 - g_n^J) v_{th}^n n + g_n^J \frac{J_n}{q} \right] \quad (372)$$

and similarly, the hole capture rate from the valence band is $c_V^p = \sigma_p \left[(1 - g_p^J) v_{th}^p p + g_p^J \frac{J_p}{q} \right]$.

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 zero (the default), the V-model is obtained, and when 1, the J-model (popular for modeling radiation problems) is obtained.

The electron emission rate to the conduction band is $e_C^n = v_{th}^n \sigma_n \gamma_n n_1 / g_n + e_{const}^n$ and the hole emission to the valence band is $e_V^p = v_{th}^p \sigma_p \gamma_p p_1 / g_p + e_{const}^p$. For $g_n^J = e_{const}^n = 0$ and $g_p^J = e_{const}^p = 0$, these rates obey the principle of detailed balance.

Above, $n_1 = n_{i,eff} \exp(E_{trap}/kT)$ and $p_1 = n_{i,eff} \exp(-E_{trap}/kT)$. For Fermi statistics or with quantization (see [Chapter 7 on page 251](#)), γ_n and γ_p are given by [Eq. 90, p. 202](#) and [Eq. 91, p. 202](#) (see [Fermi Statistics on page 202](#)), whereas otherwise, $\gamma_n = \gamma_p = 1$.

v_{th}^n and v_{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):

$$v_{th}^{n,p} = \begin{cases} v_o^{n,p} \sqrt{\frac{T}{300\text{K}}} & \text{VthFormula=1} \\ \sqrt{\frac{3kT}{m_{n,p}(300\text{K})}} & \text{VthFormula=2} \end{cases} \quad (373)$$

$v_o^{n,p}$ are given by the `Vth` parameter pair in the `Traps` section of the parameter file.

Sentaurus Device offers several options for the cross sections σ_n and σ_p . They are selected by keywords in the command file or by the value of the `XsecFormula` parameter pair in the `Traps` parameter set. In any case, trap cross sections are derived from the constant cross sections σ_n^0 and σ_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_{const}^n and e_{const}^p represent the constant emission rate terms for carrier emission from the trap level to the conduction and valence band. They can be set by the eConstEmissionRate and hConstEmissionRate keywords in the command file, and by the ConstEmissionRate parameter pair in the Traps parameter set (default $e_{\text{const}}^n = 0 \text{ s}^{-1}$, $e_{\text{const}}^p = 0 \text{ s}^{-1}$).

J-Model Cross Sections

This model is activated by the ElectricField keyword in the command file or by a value of 2 in the XsecFormula parameter pair in the Traps parameter set. It is intended for use with the J-model and reads:

$$\sigma_{n,p} = \sigma_{n,p}^0 \left(1 + a_1 \left| \frac{F}{1 \text{ Vm}^{-1}} \right|^{p_1} + a_2 \left| \frac{F}{1 \text{ Vm}^{-1}} \right|^{p_2} \right)^{p_0} \quad (374)$$

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.

Hurkx Model for Cross Sections

The Hurkx model is selected by the Tunneling (Hurkx) keyword in the command file or by a value of 3 in the XsecFormula parameter pair in the Traps parameter set. The cross sections are obtained from σ_n^0 and σ_p^0 using Eq. 293, p. 316.

Poole–Frenkel Model for Cross Sections

The Poole–Frenkel model [1] is frequently used for the interpretation of transport effects in dielectrics and amorphous films. The model predicts an enhanced emission probability Γ_{pf} for charged trap centers where the potential barrier is reduced because of the high external electric field.

The model is selected by the PooleFrenke1 keyword in the command file or by a value of 4 in the XsecFormula parameter pair in the Traps parameter set. In the Poole–Frenkel model:

$$\begin{aligned} \sigma_{n,p}^{\text{enh}} &= \sigma_{n,p}^0 (1 + \Gamma_{\text{pf}}) \\ \Gamma_{\text{pf}} &= \frac{1}{a^2} [1 + (\alpha - 1) \exp(\alpha)] - \frac{1}{2} \\ \alpha &= \frac{1}{kT} \sqrt{\frac{q^3 F}{\pi \epsilon_{\text{pf}}}} \end{aligned} \quad (375)$$

10: Traps and Fixed Charges

Trap Models and Parameters

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 $\sigma_n = \sigma_n^0$. ε_{pf} is an adjustable parameter and is set by the `epsPF` parameter pair in the PooleFrenkel parameter set.

Local Capture and Emission Rates Based on Makram-Ebeid–Lannoo Phonon-assisted Tunnel Ionization Model

The Makram-Ebeid–Lannoo model [2] 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 [3].

In this model, the deep trap acts as an oscillator or core embedded in the nitride lattice, attracting electrons (electron trap) or holes (hole trap). The deep trap is defined by the phonon energy W_{ph} , the thermal energy W_T and the optical energy W_{opt} . The trap ionization rate is given by:

$$P = \sum_{n=-\infty}^{\infty} \exp\left[\frac{nW_{\text{ph}}}{2kT} - S \coth\frac{W_{\text{ph}}}{2kT}\right] I_n\left(\frac{S}{\sinh(W_{\text{ph}}/(2kT))}\right) P_i(W_T + nW_{\text{ph}}) \quad (376)$$

$$P_i(W) = \frac{eF}{2\sqrt{2mW}} \exp\left(-\frac{4\sqrt{2m}}{3\hbar eF} W^{3/2}\right) \quad (377)$$

where I_n is the modified Bessel function of the order n , $S = (W_{\text{opt}} - W_T)/W_{\text{ph}}$, and $P_i(W)$ is the tunnel escape rate through the triangle barrier of height W .

Sentaurus Device implements the Makram-Ebeid–Lannoo model for Level traps, eNeutral and hNeutral. 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 Eq. 376 with the corresponding W_{ph} , W_T and W_{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: $c_C^n = \sigma_n^0 [(1 - g_n^J) v_{\text{th}}^n n + g_n^J J_n^p / q]$, $c_V^p = \sigma_p^0 [(1 - g_p^J) v_{\text{th}}^p p + g_p^J J_p^p / q]$.

The model is selected by the keyword `Makram-Ebeid_Lannoo` in the command file of the trap parameter set. When the `Makram-Ebeid_Lannoo` keyword with no options is used, the trap couples to both the conduction and valence bands through Makram-Ebeid–Lannoo trap emission rates. When `Makram-Ebeid_Lannoo` is used with the option `electron` in parentheses, 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. Similarly, `Makram-Ebeid_Lannoo` with `hole` in parentheses couples an hNeutral trap to the valence

band through the Makram-Ebeid-Lannoo emission rate and to the conduction band using the default emission rate.

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, by default, computed from the corresponding emission rates using the detailed balance principle. By using the keyword `simpleCapt` in parentheses as an option for `Makram-Ebeid_Lannoo`, capture rates are computed as $c_C^n = \sigma_n \left[(1 - g_n^J) v_{th}^n n + g_n^J \frac{n}{q} \right]$ for exchange with the conduction band and $c_V^p = \sigma_p \left[(1 - g_p^J) v_{th}^p p + g_p^J J_p / q \right]$ for exchange with the valence band, where 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 zero or not specified (the default), the V-model is obtained 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 1, the J-model is obtained.

The deep trap parameters W_{ph} , W_T , and W_{opt} can be adjusted in the `Makram-Ebeid_Lannoo` section of the parameter file. Their default values are $W_{ph} = 0.06\text{ eV}$, $W_T = 1.4\text{ eV}$, and $W_{opt} = 2.8\text{ eV}$. In addition, the Makram-Ebeid-Lannoo tunneling masses m can be adjusted in the parameter file. The default value is 0.5 for both electrons and holes.

Local Capture and Emission Rates from PMI

As an alternative to models described above, the local capture and emission rates c_C^n , c_V^p , e_C^n , and e_V^p can be computed directly using a physical model interface (PMI). Using this PMI only makes sense for Level traps. For more details, see [Trap Capture and Emission Rates on page 1008](#).

Tunneling and 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 [4][5]. To use the nonlocal tunneling model for tunneling to traps:

- Generate nonlocal meshes that connect the trap locations with the interfaces as required (see [Defining Nonlocal Meshes on page 518](#)).
- Specify `eBarrierTunneling` (for coupling to the conduction band) and `hBarrierTunneling` (for coupling to the valence band) in the trap specification in the command file.
- Adjust `TrapVolume`, `HuangRhys`, and `PhononEnergy` (see below), as well as the tunneling masses and the interface-specific prefactors (see [Nonlocal Tunneling Parameters on page 521](#)).

10: Traps and Fixed Charges

Trap Models and Parameters

The electron capture rate for the phonon-assisted transition from the conduction band is:

$$c_{C,\text{phonon}}^n = \frac{\sqrt{m_t m_0^3 k^3 T_n^3} g_C}{\hbar^3 \sqrt{\chi}} V_T S^2 \omega \left(1 - \frac{l}{S}\right)^2 \exp \left[-S(f_B + 1) + \frac{\Delta E}{2kT} + \chi \right] \\ \times \left(\frac{z}{l + \chi}\right)^l F_{1/2} \left(\frac{E_{F,n} - E_C(0)}{kT_n}\right) \frac{|\Psi(z_o)|^2}{|\Psi(0)|^2} \quad (378)$$

where V_T is the interaction volume of the trap, S is the Huang–Rhys factor, and $\hbar\omega$ is the energy of the phonons involved in the transition. These parameters are set by `TrapVolume` (in μm^3), `HuangRhys` (dimensionless), and `PhononEnergy` (in eV) in the command file, and by parameters of the same name in the `Traps` parameter set. All three parameters default to 0. `TrapVolume` must be positive when tunneling is activated.

In Eq. 378, 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, $z = 2S\sqrt{f_B(f_B + 1)}$ and $\chi = \sqrt{l^2 + z^2}$. The dissipated energy is $\Delta E = E_C + 3kT_n/2 - E_{\text{trap}}$. The Fermi energy and the electron temperature 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 effective tunneling mass and g_C is the prefactor for the Richardson constant at the interface or contact. See [Nonlocal Tunneling Parameters on page 521](#) for more details regarding these parameters.

The electron capture rate for the elastic transition from the conduction band is:

$$c_{C,\text{elastic}}^n = \frac{\sqrt{8m_t m_0^{3/2} g_C}}{\hbar^4 \pi} V_T [E_C(z_o) - E_{\text{trap}}]^2 \Theta[E_{\text{trap}} - E_C(0)] \sqrt{E_{\text{trap}} - E_C(0)} \left(\frac{E_{F,n} - E_{\text{trap}}}{kT_n}\right) \frac{|\Psi(z_o)|^2}{|\Psi(0)|^2} \quad (379)$$

The emission rates are obtained from the capture rates by the principle of detailed balance (see Eq. 371). The hole terms are analogous to the electrons terms. However, `hBarrierTunneling` for contacts and metals is unphysical and, therefore, ignored.

The ratio of wavefunction is obtained from Γ_{CC} described in [WKB Tunneling Probability on page 524](#) as:

$$\frac{|\Psi(z_o)|^2}{|\Psi(0)|^2} = \frac{v(0)}{v(z_0)} \Gamma_{CC} \quad (380)$$

where v denotes the (possibly imaginary) velocities. To avoid singularities, for the inelastic transition with the WKB tunneling model, the velocity $v(z_o)$ at the trap site is replaced by the thermal velocity. For the inelastic process, Γ_{CC} is computed at the tunneling energy $E_C + kT_n/2$ and, for elastic process, at energy E_{trap} .

Trap Numeric Parameters

When used with Fermi statistics, the trap model sometimes leads to convergence problems, especially at the beginning of a simulation when Sentaurus Device tries to find an initial solution. This problem can often be solved by changing the numeric damping of the trap charge in the nonlinear Poisson equation.

To this end, set the Damping option to the Traps keyword in the global Math section to a nonnegative number, for example:

```
Math {  
    Traps (Damping=100)  
}
```

Larger values of Damping increase damping of the trap charge; a value of 0 disables damping. The default value is 10. Depending on the particular example, increasing damping can improve or degrade the convergence behavior. There are no guidelines regarding the optimal value of this parameter.

At nonheterointerface vertices, bulk traps are considered only from the region with the lowest band gap. In cases where the bulk traps from other regions are important, use RegionWiseAssembly in the global Math-Traps section, which properly considers bulk traps from all adjacent regions.

Visualizing Traps

To plot the concentration of electrons trapped in eNeutral and Acceptor traps, specify eTrappedCharge in the Plot section. Similarly, for the concentration of holes trapped in hNeutral or Donor traps, specify hTrappedCharge. These datasets include the contribution of interface charges as well. To that end, Sentaurus Device converts the interface densities to volume densities and, therefore, their contribution depends on the mesh spacing. To plot the interface charges separately as interface densities, use eInterfaceTrappedCharge and hInterfaceTrappedCharge.

For backward compatibility, for traps specified in insulators, at vertices on interfaces to semiconductors, the charge density in the insulator parts associated with the vertices will be reassigned to the semiconductor parts. This involves the rescaling of the densities with the ratio of the volumes of the two parts and, typically, this leads to a distortion of the data for visualization. However, the distortion has no effect on the actual solution.

10: Traps and Fixed Charges

Visualizing Traps

To plot the recombination rates for the conduction band and valence band due to trapping and de-trapping, specify `eGapStatesRecombination` and `hGapStatesRecombination`, respectively.

In addition, Sentaurus Device allows you to plot trapped carrier density and occupancy probability versus 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 `TrappedCarPlotFile` keyword:

```
File {
    ...
    TrappedCarPlotFile = "itrapstraps_trappedcar"
}
```

The 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 defining where the trapped carrier density, occupancy probability, and trap density versus energies are to be plotted are specified materialwise or regionwise in the `TrappedCarDistrPlot` section, grouped on regions, materials, region interfaces, and material interfaces.

A set of coordinates of positions in parentheses follows the region or region interface:

```
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 the 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 closest vertex is used in distribution plotting.

Based on trap types and positions, a unique entry is created in the generated graph. The naming is `TrapTypeCounter(position)`, where `Counter` is used when multiple traps of the same type are used. For each of these entries, the `Energy`, `TrappedChargeDistribution`, `DistributionFunction`, and `TrapDensity` fields are available for plotting. In addition, for each entry, `eQuasiFermi` and `hQuasiFermi` are available for plotting. This allows you to monitor the evolution of `DistributionFunction` relative to quasi-Fermi levels.

Explicit Trap Occupation

For the investigation of, for example, time-delay effects, it may be advantageous to start a transient simulation from an initial state with either totally empty or totally filled trap states. However, depending on the position of the equilibrium Fermi level, it may be impossible to reach such an initial state from steady-state or quasistationary simulations (for example, it may be a metastable state with a very long lifetime).

For this purpose, two different mechanisms are available to initialize trap occupancies in the `Solve` section.

The first mechanism is invoked by `TrapFilling` in the `Set` and `Unset` statements of the `Solve` section. [Table 143 on page 1108](#) summarizes the syntax of these statements, and [Trap Examples on page 362](#) presents an example.

The second mechanism, invoked by `Traps` in a `Set` statement within a `Solve` section, allows you to set trap occupancies for individual named traps to spatial- and solution-independent values, and to freeze and unfreeze the trap occupancies of all traps. To set a trap you need to define an identifying `Name` in its definition in `Traps`, which enables you to reference this trap in the `Set` command. The `Frozen` option allows you to freeze the trap occupancies for subsequent `Solve` statements and to unfreeze them again. 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. Observe that `Frozen` applies to all defined traps. Traps not explicitly initialized in the `Set` are frozen at their actual values. The options of `Traps` in `Set` are summarized in [Table 143 on page 1108](#).

10: Traps and Fixed Charges

Trap Examples

```
Device "MOS" {
    Physics { ...
        Traps ( ( Name="t1" eNeutral ... ) ( Name="t2" hNeutral ... ) ... )
    }
}
System { ...
    MOS "mos1" ( ... ) { ... }
}
Solve { ...
    Set ( Traps ( "mos1"."t1" = 1. Frozen ) )
    ...
    Set ( Traps ( -Frozen ) )
    ...
}
```

Here in the example, some traps are named. In the first `Set`, the 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` releases all traps again.

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 369](#)).

Trap Examples

The following example of a trap statement 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)
```

This example shows trap specifications appropriate for a polysilicon TFT, with four exponential distributions:

```
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) )
```

The following `Solve` statement fills traps consistent with a high electron and a low hole concentration; performs a `Quasistationary`, keeping that trap filling; and, finally, simulates the transient evolution of the trap occupation:

```
Solve{
    Set(TrapFilling=n)
    Quasistationary{...}
    Unset(TrapFilling)
    Transient{...}
}
```

Insulator Fixed Charges

Sentaurus Device supports a special syntax for fixed charges in insulators and at insulator interfaces. Insulator fixed charges are defined as options to `Charge` in the `Physics` section:

```
Physics (Material="Oxide"){
    Charge (
        (<charge specification>
        (<charge specification>
    )
}
```

and likewise for `Region`, `RegionInterface`, and `MaterialInterface`. When only a single specification is present, the inner pair of parentheses can be omitted.

For bulk insulator charges, the only relevant parameter is `Conc`, the concentration, specified in $q\text{cm}^{-3}$.

Fixed charges at interfaces can be specified with either Gaussian or uniform distributions:

```
Charge([Uniform | Gaussian]
    Conc = <float>      # [cm-2]
    SpaceMid = <vector>   # [um]
    SpaceSig = <vector>   # [um]
```

The parameter `Conc` specifies the maximum surface charge concentration σ_0 , in $q\text{cm}^{-2}$. The parameters `SpaceMid` and `SpaceSig` specify the vectors (x_0, y_0, z_0) and $(\sigma_x, \sigma_y, \sigma_z)$, respectively, and are given in micrometers. They are optional for `Uniform` distributions and mandatory for `Gaussian` distributions.

10: Traps and Fixed Charges

References

For Uniform distributions (the default), if SpaceMid and SpaceSig are not specified, the surface charge concentration σ is constant on the interface, $\sigma = \sigma_0$. Otherwise, for a point (x, y, z) on the interface:

$$\sigma(x, y, z) = \sigma_0 \Theta(\sigma_x - |x - x_0|) \Theta(\sigma_y - |y - y_0|) \Theta(\sigma_z - |z - z_0|) \quad (381)$$

where Θ is the unit step function.

For Gaussian distributions, for a point (x, y, z) on the interface:

$$\sigma(x, y, z) = \sigma_0 \exp\left(-\frac{(x - x_0)^2}{2\sigma_x^2} - \frac{(y - y_0)^2}{2\sigma_y^2} - \frac{(z - z_0)^2}{2\sigma_z^2}\right) \quad (382)$$

For example, the syntax allows the specification of a piecewise constant interface charge distribution:

```
Charge ((Uniform Conc=-1.e12 SpaceMid=(0.02,0) SpaceSig=(0.01,0)
          (Gaussian Conc=2.e12 SpaceMid=(0.04,0) SpaceSig=(0.01,0)))
```

[Table 205 on page 1165](#) summarizes the options that can appear in the charge specification.

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. 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.
- [3] 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.
- [4] 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.
- [5] 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.

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 10 on page 349](#). 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 [Hydrogen Transport Degradation Model on page 381](#)), diffusing mobile hydrogen species may 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 may 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:

$$\sum_i s_i = 1 \quad (383)$$

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:

$$\dot{s}_i = \sum_{j \neq i} \sum_{t \in T_{ij}} c_{ij} s_j - e_{ij} s_i \quad (384)$$

11: Phase and State Transitions

Multistate Configurations and Their Dynamic

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:

$$\dot{s} = Ts \quad (385)$$

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:

$$K_i - K_j = \sum_r P_r K_r \quad (386)$$

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 71](#) 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 384](#).

Table 71 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

The resulting space charge and recombination terms with the corresponding equations are taken into account automatically.

Specifying Multistate Configurations

A multistate configuration is specified by an `MSConfig` section placed into an `MSConfigs` section of a (region or material or global) `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, for example:

```
Physics ( Region = "si" ) {
    MSConfigs (
        MSConfig ( Name = "ca"
            State ( Name = "c" ) State ( Name = "a" )
            Transition ( Name = "t1"
                To="c" From="a" CEModel("CEModel_ArrheniusLaw" 0))
            )
        ...
    )
}
```

This example specifies a multistate configuration with two states and one transition between them. See [Table 215 on page 1172](#) 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 trap capture and emission PMI model (see [Trap Capture and Emission Rates on page 1008](#)) including its optional model index. 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 charged 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.

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)))
```

11: Phase and State Transitions

Interaction of Multistate Configurations with Transport

```
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 [ACCoupled: Small-Signal AC Analysis on page 161](#)), harmonic balance analysis (see [Harmonic Balance on page 164](#)), and noise analysis (see [Chapter 16 on page 499](#)) do not support multistate configurations.

NOTE The computation of state occupation is influenced by the `TrapFilling` option of the `Set` and `Unset` commands in the `Solve` section (see [Table 128 on page 1094](#)). It is frozen if the trap-filling option `Frozen` is set; otherwise, the occupation rates are treated as free. The frozen status is released again by the `Unset(TrapFilling)` command.

The transition dynamic may 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 230 on page 1181](#)) in the `Plot` section.

Interaction of Multistate Configurations with Transport

The 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 a user-defined `PMI_MSC_ApparentBandEdgeShift` model described in [Multistate Configuration-dependent Apparent Band-Edge Shift on page 973](#).

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 }
    }
}
```

Thermal Conductivity, Heat Capacity, and Mobility

Descriptions of PMIs for MSC-dependent thermal conductivity, heat capacity, and mobility can be found in [Multistate Configuration–dependent Thermal Conductivity on page 991](#), [Multistate Configuration–dependent Heat Capacity on page 996](#), and [Multistate Configuration–dependent Bulk Mobility on page 947](#), respectively.

Manipulating MSCs During Solve

The MSC dynamic is, in general, solved implicitly. However, sometimes it may be useful to manipulate the computations. For example, you may 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 may 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.

11: Phase and State Transitions

Manipulating MSCs During Solve

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 137 on page 1104](#) 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
        )
    )
    ...
    Set ( MSConfigs ( -Frozen ) )           # unfreeze the MSC dynamic
}
```

Here, the occupancies of states `s0` and `s1` 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 361](#)).

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 $\tilde{c} = \kappa c$ (κ 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.

The PCM device is modeled by a two-state model representing the crystalline and amorphous phase using a multistate configuration with two states. The transition is modeled by the temperature-dependent capture and emission rate model `CEModel_ArrheniusLaw` (see [Example: CEModel_ArrheniusLaw on page 1011](#)). The model has four parameters:

- The energy difference δE and degeneracy factor g determine the equilibrium.
- The activation energy E_{act} and r_0 determine the velocity of the transition.

11: Phase and State Transitions

Example: Two-State Phase-Change Memory Model

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_{act} is the activation energy.

This chapter discusses the degradation models used in Sentaurus Device.

Overview

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. Sentaurus Device provides two degradation models:

- Trap degradation model
 - Hydrogen transport degradation model
-

Trap Degradation Model

Disorder-induced variations among the Si-H activation energies at the passivated Si–SiO₂ interface have been shown [1] 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 [2]. In addition, this could be due to a Si-H density-dependent activation energy [3], which may 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–SiO₂ interface were passivated by hydrogen (H) or deuterium (D) [4], 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 [5] 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.

Assuming $N = N_{hb}^0 + N_{it}^0$ total Si bonds at the interface, the remaining number of Si-H bonds at the interface after stress is $N_{hb} = N - N_{it}$ and follows the power law:

$$N_{hb} = \frac{N_{hb}^0}{1 + (vt)^{\alpha}} \quad (387)$$

Based on experimental observations, the power α is stress dependent and varies between 0 and 1.

From first-order kinetics [1], it is expected that the Si-H concentration during stress obeys:

$$\frac{dN_{hb}}{dt} = -vN_{hb} \quad (388)$$

where v is a reaction constant that can be described by $v \propto v_A \exp(-\varepsilon_A/kT)$ in the Arrhenius approximation, ε_A is the Si-H activation energy, and T is the Si-H temperature. The exponential kinetics given by this equation ($N_{hb} = N_{hb}^0 \exp(-vt)$) do not fully describe the experimental data because a constant activation energy will behave like the power law in Eq. 387, 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 [6]. 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 $N - N_{hb}$, so the activation energy dependence (assuming the activation energy changes logarithmically with the breaking of Si-H) can be expressed as:

$$\varepsilon_A = \varepsilon_A^0 + (1 + \beta)kT \ln \left(\frac{N - N_{hb}}{N - N_{hb}^0} \right) \quad (389)$$

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 α 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 [7]. 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:

$$\begin{aligned} D \frac{dN_H}{dx} &= \frac{dN_{\text{hb}}}{dt} & x = 0 \\ \frac{dN_H}{dt} &= D \frac{d^2N_H}{dx^2} & 0 < x < x_P \\ D \frac{dN_H}{dx} &= -k_P(N_H - N_H^0) & x = x_P \end{aligned} \quad (390)$$

where N_H is a concentration of hydrogen in oxide, $D = D_0 \exp(-\varepsilon_H/kT)$ 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.

Syntax and Parameterized Equations

The general kinetic equation (left column of [Eq. 391](#)) with an added passivation term can be activated by the keyword `Degradation`. The power law (right column of [Eq. 391](#)) is activated by the keyword `Degradation(PowerLaw)`:

Kinetic	Power Law
$\frac{dN_{\text{hb}}}{dt} = -vN_{\text{hb}} + \gamma(N - N_{\text{hb}})$	$N_{\text{hb}} = \frac{N_{\text{hb}}^0}{1 + (vt)^\alpha}$
$\gamma = \gamma_0[N_{\text{H}}/N_{\text{H}}^0 + \Omega(N_{\text{hb}}^0 - N_{\text{hb}})]$	$\alpha = 0.5 + \beta$
$\gamma_0 = \frac{N_{\text{hb}}^0}{N - N_{\text{hb}}^0} v_0$	

(391)

where γ_0 is the passivation constant, which is computed automatically by default, to provide the equilibrium, but you can specify it directly in the input file. Ω is the passivation volume (by default, it is equal to zero), and it represents a simple model of the effect that depassivated hydrogen can be trapped by dangling silicon bonds again. 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 375](#)). If the R-D model is not activated, then $N_{\text{H}}/N_{\text{H}}^0 = 1$.

The degradation model can be activated for any trap level or distribution (see [Chapter 10 on page 349](#)). Its keywords are described in [Table 224 on page 1176](#) (see the options referred to in [Chapter 12 on page 373](#)) and can be used with any other trap options listed in the same table. The model applied to the trap distribution $N(\varepsilon)$ controls the total trap concentration $\int_{E_V}^{E_C} N(\varepsilon) d\varepsilon$.

The parameterized system of equations for the reaction constant v based on the trap formation model (see [Trap Formation Kinetics on page 373](#)) and [3] can be expressed as:

$$\begin{aligned}
 v &= v_0 \exp\left(\frac{\varepsilon_A^0}{kT_0} - \frac{\varepsilon_A^0 + \Delta\varepsilon_A}{\varepsilon_T}\right) k_{\text{FN}} k_{\text{HC}} \\
 \varepsilon_T &= kT + \delta_{\parallel} |F_{\parallel}|^{\rho_{\parallel}} \\
 k_{\text{HC}} &= 1 + \delta_{\text{HC}} |I_{\text{HC}}|^{\rho_{\text{HC}}} \\
 k_{\text{FN}} &= 1 + \delta_{\text{Tun}} |I_{\text{Tun}}|^{\rho_{\text{Tun}}} \\
 \Delta\varepsilon_A &= -\delta_{\perp} |F_{\perp}|^{\rho_{\perp}} + (1 + \beta) \varepsilon_T \ln \frac{N - N_{\text{hb}}}{N - N_{\text{hb}}^0} \\
 \beta &= \beta_0 + \beta_{\perp} F_{\perp} + \beta_{\parallel} F_{\parallel}
 \end{aligned} \tag{392}$$

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_{\parallel} are 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. I_{HC}, I_{Tun} are the local densities of hot carrier and tunneling (Fowler–Nordheim and direct) currents at the interface where traps are located. These currents represent hot carrier (see [Chapter 18 on page 531](#)) and tunneling stresses (see [Chapter 17 on page 511](#)) and should be activated in the GateCurrent model.

ε_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 $\delta_{\parallel}|F_{\parallel}|^{\rho_{\parallel}}$. $\Delta\epsilon_A$ is a change of the activation energy because of stretched Si-H bonds [8] by the electric field (first term) and due to a change of the chemical potential (second term) [3]. Effectively, the influence of the chemical potential also can be different in the presence of the electric field, and the coefficient β represents this. Coefficients $\delta_{\perp}, \rho_{\perp}, \delta_{\parallel}, \rho_{\parallel}, \delta_{HC}, \rho_{HC}, \delta_{Tun}, \rho_{Tun}$, and $\beta_0, \beta_{\perp}, \beta_{\parallel}$ are field and current enhancement parameters of the model. [Table 224 on page 1176](#) lists the options available for the degradation model.

NOTE [Eq. 390](#), [Eq. 391](#), and [Eq. 392](#) are designed to represent degradation of the interface traps, but you can use it for bulk trap degradation as well. For the bulk traps, Sentaurus Device will set $F_{\perp} = F$ and $F_{\parallel} = I_{HC} = I_{Tun} = 0$, but you should carefully reconsider all parameters of the model.

When the tail-electron energy distribution is available by specifying the keyword TaileDistribution or TailDistribution in the GateCurrent statement (see [Tail Distribution Hot-Carrier Injection on page 537](#)), you can include the following additional tail distribution enhancement factor:

$$k_{tail} = 1 + \delta_{tail} \frac{q}{4} \int_{\varepsilon_{th}}^{\infty} \left(\min \left[\exp \left(\frac{\varepsilon - \varepsilon_a}{kT} \right), 1 \right] g(\varepsilon) f(\varepsilon) v(\varepsilon) \right) d\varepsilon \quad (393)$$

where δ_{tail} is a prefactor, ε_{th} is a threshold energy, ε_a is an activation energy, g is the density-of-states, f is the electron energy distribution, and v is the magnitude of the electron velocity. This enhancement factor is multiplied by the reaction coefficient. The tail distribution enhancement factor can be specified by TaileDistribution keyword in the Traps statement. Similarly, the TailhDistribution keyword specifies the enhancement factor of the tail-hole distribution.

12: Degradation Model

Trap Degradation Model

The following example specifies the tail distribution enhancement factor with $\delta_{\text{tail}} = 100 \text{ cm}^2/\text{A}$, $\varepsilon_{\text{th}} = 2 \text{ eV}$, and $\varepsilon_a = 2.1 \text{ eV}$:

```
Physics(MaterialInterface="Silicon/Oxide") {
    Traps(... {
        Degradation
        TaileDistribution=(1.0e2 2 2.1) # (delta_tail E_th, E_a)
        ...
    })
}
```

Device Lifetime and Simulation

Based on [Syntax and Parameterized Equations on page 376](#), the Physics section of an input file that describes the parameters of the degradation model can be:

```
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 cm^{-2} and, in the process of degradation, it can be increased up to 10^{12} 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 }
}
}

```

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 `TotalInterfaceTrapConcentration`, `TotalTrapConcentration`, and `OneOverDegradationTime`, for example:

```

CurrentPlot{
    eDensity(359) Potential(359)
    eInterfaceTrappedCharge(359) hInterfaceTrappedCharge(359)
    OneOverDegradationTime(359) TotalInterfaceTrapConcentration(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_{crit} , which defines an edge between a properly working and improperly working device. Using N_{crit} , the device lifetime τ_D is defined as follows (according to the two different prediction ways):

1. In transient, direct computation of time $t = \tau_D$ gives the trap concentration equal to N_{crit} .
2. In Quasistationary, if the previously finished transient statement computes the device lifetime τ_D^{stress} and the depassivation constant v^{stress} at stress conditions, then $\tau_D = (v^{\text{stress}}/v)\tau_D^{\text{stress}}$.

12: Degradation Model

Trap Degradation Model

So, the plotted value of OneOverDegradationTime is equal to $1/\tau_D$ for one trap level and the sum $\sum 1/\tau_D^i$ 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.

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 may 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.

Hydrogen Transport Degradation Model

Oxide degradation caused by bias and temperature stress is believed to be related to the silicon-hydrogen bond depassivation and the subsequent hydrogen transport [9][10][11]. This model provides a general framework for hydrogen transport-related degradation models. In the model, you can specify:

- 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 11 on page 365](#)).
-

Hydrogen Transport

Transport equations for hydrogen atoms (X_1), hydrogen molecules (X_2), and hydrogen ions (X_3) can be written as:

$$\frac{\partial}{\partial t}[X_i] + \nabla \cdot \left[D_i \exp\left(-\frac{E_{di}}{kT}\right) \left(\frac{qK_i^Q}{kT} \vec{F}[X_i] - \nabla[X_i] \right) \right] + R_{\text{net}} + r_i([X_i] - [X_i]_0) = 0 \quad (394)$$

where D_i is the diffusion coefficient, E_{di} is the diffusion activation energy, K_i^Q is the number of charges for element X_i , R_{net} is the net recombination rate due to chemical reactions, r_i is the explicit recombination rate, and $[X_i]_0$ is the reference density. At electrodes, $[X_i] = [X_i]_0$ is assumed as a boundary condition.

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 α_i and β_i are the particle numbers of element X_i to be removed and created by the forward reaction.

12: Degradation Model

Hydrogen Transport Degradation Model

These coefficients must satisfy the charge and hydrogen conservation laws:

$$\sum_{i=1}^5 (\alpha_i - \beta_i) K_i^Q = 0 \quad (396)$$

$$\sum_{i=1}^5 (\alpha_i - \beta_i) K_i^H = 0 \quad (397)$$

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 71 on page 366](#)).

The forward and reverse reaction rates R_f and R_r are modeled as:

$$R_f = k_f \exp\left(\delta_f F - \frac{E_f}{kT}\right) \prod_{i=1}^5 \left(\frac{[X_i]}{1 \text{ cm}^3}\right)^{\alpha_i} \quad (398)$$

$$R_r = k_r \exp\left(\delta_r F - \frac{E_r}{kT}\right) \prod_{i=1}^5 \left(\frac{[X_i]}{1 \text{ cm}^3}\right)^{\beta_i} \quad (399)$$

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).
- δ_f and δ_r are the forward and reverse reaction field coefficients, respectively cm V^{-1} .
- E_f and E_r are the forward and reverse reaction activation energies, respectively [eV].

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 input command file can be written as:

```

Physics ( Material = "Oxide" ) {
    HydrogenDiffusion(
        HydrogenReaction( # 2H <-> 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 210 on page 1167](#) 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"
        )
    )
}

```

12: Degradation Model

Hydrogen Transport Degradation Model

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⁺, 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_HydrogenDepassivation")
                Reservoirs("VB"(Particles=+1) "HydrogenAtom"(Particles=-1) )
            )
        )
        ...
    )
}
```

The capture and emission rates are obtained from:

$$c_{21} = c_{\text{PMI}}(n, p, T, T_n, T_p, F) \prod_{i=1}^3 \left(\frac{[X_i]}{1/\text{cm}^3} \right)^{\alpha_i} \quad (401)$$

$$e_{21} = e_{\text{PMI}}(n, p, T, T_n, T_p, F) \prod_{i=1}^3 \left(\frac{[X_i]}{1/\text{cm}^3} \right)^{\beta_i} \quad (402)$$

where c_{PMI} and e_{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 11 on page 365](#).

NOTE Contrary to the conventional trap capture and emission PMI model, the insulator electric field is used in the PMI model at the semiconductor–insulator interface instead of the semiconductor electric field when the hydrogen elements are involved in the transition.

Using 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 input 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 381](#)).

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`, and `HydrogenIon` must be specified inside the `Coupled` command 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]
            n0 = 0       # [cm^-3]
            krec = 0     # [cm^-3*s^-1]
        }
        HydrogenMolecule {
            d0 = 1.0e-14 # [cm^2*s^-1]
            Ed = 0       # [eV]
```

12: Degradation Model

References

```
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]
        }
    }
}
```

Here d_0 , E_d , n_0 , and k_{rec} correspond to D_i , E_{di} , $[X_i]_0$, and r_i . The default values of these parameters are zero.

The keywords for plotting the densities of hydrogen atoms, hydrogen molecules, and hydrogen ions are:

```
Plot {
    HydrogenAtom HydrogenMolecule HydrogenIon
}
```

References

- [1] A. Plonka, *Time-Dependent Reactivity of Species in Condensed Media*, Lecture Notes in Chemistry, vol. 40, Berlin: Springer, 1986.
- [2] 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.
- [3] O. Penzin *et al.*, “MOSFET Degradation Kinetics and Its Simulation,” *IEEE Transactions on Electron Devices*, vol. 50, no. 6, pp. 1445–1450, 2003.
- [4] K. Hess *et al.*, “Theory of channel hot-carrier degradation in MOSFETs,” *Physica B*, vol. 272, no. 1–4, pp. 527–531, 1999.
- [5] 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.

- [6] 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.
- [7] M. A. Alam and S. Mahapatra, "A comprehensive model of PMOS NBTI degradation," *Microelectronics Reliability*, vol. 45, no. 1, pp. 71–81, 2005.
- [8] J. W. McPherson, R. B. Khamankar, and A. Shanware, "Complementary model for intrinsic time-dependent dielectric breakdown in SiO_2 dielectrics," *Journal of Applied Physics*, vol. 88, no. 9, pp. 5351–5359, 2000.
- [9] 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.
- [10] 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.
- [11] T. Grasser, W. Gös, 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.

12: Degradation Model

References

CHAPTER 13 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 models needed in a typical organic device simulation are:

- 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. Note that it is common for electrons to have two orders of magnitude higher mobility than holes (see [Poole–Frenkel Mobility \(Organic Material Mobility\) on page 303](#)).
- 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 561](#)).
- 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 240](#)).
- The traps model must be initialized with the appropriate capture cross sections and densities (see [Chapter 10 on page 349](#)).
- The Langevin bimolecular recombination model is used to model the recombination process of carriers and the generation process of singlet excitons (see [Bimolecular Recombination on page 344](#)).

13: Organic Devices

References

- 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. Note that 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 189](#)).
- For optical emissions in organic light-emitting diodes (OLEDs), the photon generation rate is derived from the radiative decay of singlet excitons (see [OLED Emissions on page 758](#)).

The organic device simulator is based on the work of Kozlowski [1], and other useful papers are available [2]–[10].

Many acronyms are used for the description of organic device layers and transport mechanisms. [Table 72](#) lists commonly used acronyms for organic transport.

Table 72 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
TSL	Thermally stimulated luminescence

Examples of simulating OLEDs can be found in [Simulating Organic Light-emitting Diode on page 676](#) and [Modeling Organic Light-emitting Diodes on page 757](#).

References

- [1] F. Kozlowski, *Numerical simulation and optimisation of organic light emitting diodes and photovoltaic cells*, Ph.D. 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.

13: Organic Devices

References

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, and the beam propagation method. Different types of refractive index and absorption models, wavelength ramping, and optical AC analysis are also described.

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 project, while only the solver-specific parameters have to change. Currently supported are the transfer matrix method, the raytracing method, and the finite-difference time-domain (FDTD) method including its hardware-accelerated version.

NOTE You are encouraged to switch to the unified interface whenever possible as it offers more functionality and its application is less error prone. However, full backward compatibility is maintained. To avoid duplication at the beginning of this chapter, which describes the unified interface for optical generation computation, references to later sections are given to explain the details of the various optical solvers.

Unified Interface for Optical Generation Computation

The unified interface for optical generation computation supports the transfer matrix method, the FDTD method including its hardware-accelerated version, and the raytracing method. 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.

14: Optical Generation

Unified Interface for Optical Generation Computation

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.

In the following sections, the different approaches and their corresponding parameters are described.

Specifying the Type of Optical Generation Computation

In the `OpticalGeneration` section of the command file, one or more methods to compute the optical generation can be specified:

- `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` enables you to set a background constant optical generation in the specified region or material.

If several methods are specified, 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 (...)  
        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
    )
)
```

Note that both `Scaling` and `TimeDependence` can also be specified directly in the `OpticalGeneration` section if the same parameters will apply to all methods. For an overview of all available options, see [Table 174 on page 1143](#).

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" ) { -OpticalGeneration }
Physics ( material="InP" ) { -OpticalGeneration }
```

To visualize the optical intensity and generation, the keywords `OpticalIntensity` and `OpticalGeneration` must be added to the `Plot` section:

```
Plot {
    ...
    OpticalIntensity
    OpticalGeneration
}
```

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

14: Optical Generation

Unified Interface for Optical Generation Computation

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 400](#) and [Setting the Excitation Parameters on page 404](#)).

Together with the possibility of ramping parameters (see [Controlling Computation of Optical Problem in Solve Section on page 405](#)), 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 ( ... )
        )
    )
}
```

The illumination spectrum file has a two-column format. The first column contains the wavelength in μm and the second column contains the intensity in W/cm^2 . The characters # and * mark the beginning of a comment.

NOTE The units of intensity are W/cm^2 in the illumination spectrum file here. This is different from the old format of specifying the illumination spectrum where the units were W/m^2 (see [Spectral Illumination on page 486](#)).

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. However, it is possible to force its recomputation on every occasion by specifying the keyword RefreshEveryTime in the ComputeFromSpectrum section.

Loading and Saving Optical Generation from and to File

Sometimes, solving the optical problem may require long computation times, in which case, it can be useful to save the solution to a file and to load the optical generation profile in later simulations whose optical properties remain constant. Currently, this feature is limited to simulations using the same grid for both saving the profile and loading it. However, using the submesh feature in an intermediate step overcomes this limitation (see the *Mesh Generation Tools User Guide* for details).

The command file syntax for saving the optical generation rate to a file is:

```
File {  
    ...  
    OpticalGenerationOutput = <filename>  
}
```

For loading the optical generation rate from a file, the syntax is:

```
File {  
    ...  
    OpticalGenerationInput = <filename>  
}  
  
Physics {  
    ...  
    Optics (  
        ...  
        OpticalGeneration (  
            ...  
            ReadFromFile ( ... )  
        )  
    )  
}
```

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 = <name of region>) {  
    ...  
    Optics (  
        OpticalGeneration (  
            SetConstant (  
                Value = <float>  
            )  
        )  
    )
```

14: Optical Generation

Unified Interface for Optical Generation Computation

```
)  
}  
  
Physics (Material = <name of material>) {  
  
...  
Optics (  
    OpticalGeneration (  
        SetConstant (  
            Value = <float>  
        )  
    )  
)  
}
```

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>  
        )  
    )  
)  
}
```

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. 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. If the optical generation needs to be scaled inside a `Quasistationary`, the keyword `Scaling` in the `OpticalGeneration` section can be used. Three types of time dependency are available:

- Linear time dependency
- Gaussian time dependency
- Arbitrary time dependency read from a file

For the linear and Gaussian time dependencies, a time interval `WaveTime = (<t1>, <t2>)` can be specified. Before `<t1>`, the optical generation rate undergoes a linear or Gaussian

increase from zero. After `<t2>`, the optical generation rate experiences a linear or Gaussian decrease.

The linear time function is expressed as:

$$F(t) = \begin{cases} \max(0, m(t - t_1) + 1) & , t < t_1 \\ 1 & , t_1 \leq t \leq t_2 \\ \max(0, m(t_2 - t) + 1) & , t > t_2 \end{cases} \quad (403)$$

where m is given by `WaveTSlope`.

The Gaussian time function is expressed as:

$$F(t) = \begin{cases} \exp\left(-\left(\frac{t_1 - t}{\sigma}\right)^2\right) & , t < t_1 \\ 1 & , t_1 \leq t \leq t_2 \\ \exp\left(-\left(\frac{t - t_2}{\sigma}\right)^2\right) & , t > t_2 \end{cases} \quad (404)$$

where σ is defined by `WaveTSigma`.

If no time interval is specified then $t_1 = t_2 = 0$ is assumed. The linear time dependency is selected by specifying the parameter `WaveTSlope`; whereas, the Gaussian time dependency is chosen by specifying the parameter `WaveTSigma`.

An arbitrary time dependency can be applied by defining a time function whose interpolation points are given in a file with a whitespace-separated, two-column format. The first column contains the time points in seconds and the second column 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 (
        ...
        OpticalGeneration (
            ...
            TimeDependence (FromFile)
        )
    )
}
```

14: Optical Generation

Unified Interface for Optical Generation Computation

```
    )  
}
```

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) method including the hardware-accelerated option
- Raytracing (RT)

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 442](#) with the only difference that the excitation parameters must be specified directly in the `Excitation` section of the `OpticalGeneration` section for all solvers. Since the excitation parameters have been unified for all optical solvers, it is possible that some units have changed compared to the original definition within a specific solver. Refer to [Setting the Excitation Parameters on page 404](#) for details about the excitation parameters.

Finite-Difference Time-Domain Method

In contrast to the TMM solver, the FDTD-specific parameters, such as boundary conditions, are defined in a separate input file that is listed in the `File` section. The available FDTD solvers are:

- Native Sentaurus Device Electromagnetic Wave Solver (EMW); no option keyword necessary.
- Three-dimensional oblique periodic boundary conditions (keyword `EMWplusOption`).
- Acceleware hardware acceleration (keyword `AccelewareOption`) (sold separately).

The syntax for activating the various FDTD solvers is similar and requires the input of the file name of the command file of the specific FDTD solver, and a keyword option to choose the specific solver. The general syntax is:

```
File {  
    ...  
    OpticalSolverInput = <command file for emw, 'emw -plus', 'emw -acceleware'>  
}  
  
Physics {  
    ...  
    Optics (...  
        OpticalSolver (  
            FDTD ( EMWplusOption )      # or AccelewareOption  
        )  
    )  
}
```

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 396](#) and [Parameter Ramping on page 406](#)).

To activate this feature, `GenerateMesh (...)` must be specified in the FDTD section and a common basename (that is, file name excluding suffix) for the boundary file and the Sentaurus Mesh command file must be defined in the `File` section. 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

14: Optical Generation

Unified Interface for Optical Generation Computation

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, refer to the *Sentaurus Device Electromagnetic Wave Solver User Guide*.

Raytracing

The raytracer-specific parameters are a smaller subset of those of the independent raytracer. There are new or modified sections added to make the raytracer more versatile and to give you more flexibility to control the raytracer parameters. 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(...)
)
}
```

where the RayTracing section contains:

```

RayTracing(
    # Retaining keywords of independent raytracer
    # -----
    CompactMemoryOption
    MonteCarlo
    OmitReflectedRays
    OmitWeakerRays
    RedistributeStoppedRays

    PolarizationVector = vector
    PolarizationVector = Random
    DepthLimit = integer
    MinIntensity = float      # fraction
    RelativeMinIntensity = float # fraction
    Print(Skip(integer))
    ProgressMarkers = integer
    VirtualRegions { "string" "string" ... }
    VirtualRegions {}

    # New/Modified keywords:
    # -----
    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 * 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)
    )
)

```

Some comments about the RT syntax:

- In the RectangularWindow definition of starting rays, you select the discretization of the window, and the position of each starting ray will be centered in each discretized cell of the window. The starting position remains as defined even if the ray direction is changed. If RayDirection is omitted, the direction is computed from Theta and Phi of the Excitation section. In this way, you can also ramp the direction of the input rays.

14: Optical Generation

Unified Interface for Optical Generation Computation

- 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, and area of each starting ray, thereby achieving greater flexibility. The power (W) of each ray is then computed by multiplying the input area (cm^2) by the input wave power (W/cm^2). Remarks are allowed but must begin with a hash (#) sign. A sample of this file is:

```
filename.txt:  
# Position(x,y,z) [um]  Direction(x,y,z) [um]  Area [cm^2]  
0.0    0.0    0.0    0.0    0.0    1.0    1.0e-5  
0.0    0.05   0.0    0.0    0.0    1.0    1.0e-5  
0.0    0.05   0.05   0.0    0.0    1.0    1.0e-5  
...  
...
```

Setting the Excitation Parameters

The Excitation section is common to all optical solvers and contains the following keywords:

- Intensity [W/cm^2]
- Wavelength [μm]
- Theta [deg]
- Phi [deg]
- PolarizationAngle [deg] or Polarization

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.

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 $z \times k$ axis, where k is the unit wavevector (see [Sentaurus Device Electromagnetic Wave Solver User Guide, Plane Wave on page 44](#)). The keyword Polarization is used for scalar methods. Possible values are TE, TM, or a real number in the interval [0, 1] in the case of the TMM solver (see [Using Transfer Matrix Method on page 442](#)) or the RT solver.

Choosing Refractive Index Model

The complex refractive index model as described in [Complex Refractive Index Model on page 477](#) is currently only available for the TMM solver and RT solver, and it has to be specified in the Optics section when using the unified interface for optical generation computation. Otherwise, the refractive index and the extinction coefficient are taken from the TableODB in the parameter file.

The FDTD solver uses the material parameter specification documented in the *Sentaurus Device Electromagnetic Wave Solver User Guide*.

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 }
```

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 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.

14: Optical Generation

Unified Interface for Optical Generation Computation

Parameter Ramping

Sentaurus Device can be used to ramp the values of physical parameters (see [Ramping Physical Parameter Values on page 71](#)). 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 { ModelParameter = "Optics/Excitation/Wavelength" Value = <float> }  
        ){ Optics }  
    }  
}
```

Similarly, the corresponding `CurrentPlot` section reads as follows:

```
CurrentPlot { ModelParameter = "Optics/Excitation/Wavelength" }
```

[Table 73](#) lists the parameters that can be ramped using the unified interface for optical generation computation.

Table 73 Parameters that can be ramped

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/OpticalSolver/TMM	NodesPerWavelength
Optics/OpticalSolver/TMM/Stripe	Left, Right

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>
```

Raytracing

Sentaurus Device supports the simulation of photogeneration by raytracing in 2D and 3D for arbitrarily shaped structures. 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 477](#)). 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 28](#).

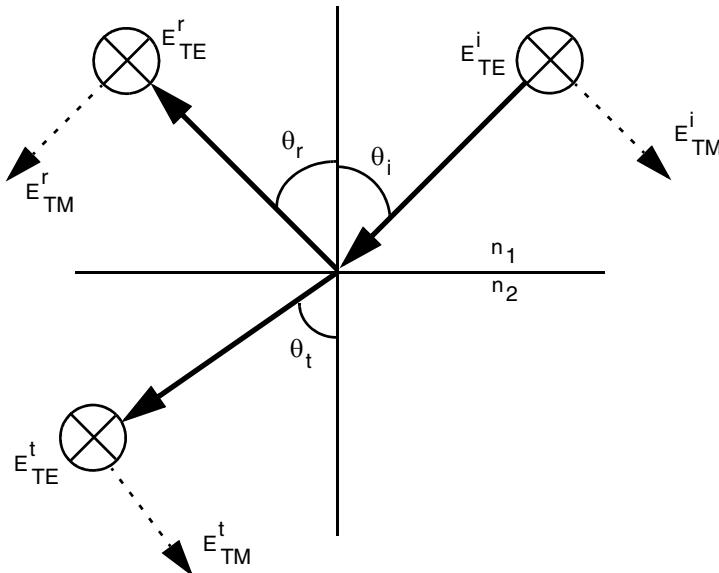


Figure 28 Incident ray splits into reflected and transmitted rays at an interface: the TE component of the polarization vector maintains the same direction, whereas the TM component changes direction

14: Optical Generation

Raytracing

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 θ_i , θ_r , and θ_t , respectively. These angles can be derived from the concept of interface tangential phase-matching (commonly called Snell's law) using:

$$n_1 \sin \theta_i = n_2 \sin \theta_t \quad (405)$$

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 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. In [Figure 28 on page 407](#), 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:

$$r_{TE} = \frac{k_{1z} - k_{2z}}{k_{1z} + k_{2z}} \quad (406)$$

$$r_{TM} = \frac{\epsilon_2 k_{1z} - \epsilon_1 k_{2z}}{\epsilon_2 k_{1z} + \epsilon_1 k_{2z}} \quad (407)$$

Amplitude transmission coefficients:

$$t_{TE} = \frac{2k_{1z}}{k_{1z} + k_{2z}} \quad (408)$$

$$t_{TM} = \frac{2\epsilon_2 k_{1z}}{\epsilon_2 k_{1z} + \epsilon_1 k_{2z}} \quad (409)$$

Power reflection coefficients:

$$R_{\text{TE}} = |r_{\text{TE}}|^2 \quad (410)$$

$$R_{\text{TM}} = |r_{\text{TM}}|^2 \quad (411)$$

Power transmission coefficients:

$$T_{\text{TE}} = \frac{k_{2z}}{k_{1z}} |t_{\text{TE}}|^2 \quad (412)$$

$$T_{\text{TM}} = \frac{\epsilon_1 k_{2z}}{\epsilon_2 k_{1z}} |t_{\text{TM}}|^2 \quad (413)$$

where:

$$k_0 = 2\pi/\lambda_0 \quad (414)$$

$$k_{1z} = n_1 k_0 \cos \theta_i \quad (415)$$

$$k_{2z} = n_2 k_0 \cos \theta_t \quad (416)$$

$$\epsilon_1 = n_1^2 \quad (417)$$

$$\epsilon_2 = n_2^2 \quad (418)$$

where λ_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 $\theta_i = \theta_t = 0$, $r_{\text{TE}} = -r_{\text{TM}}$, and $R_{\text{TE}} = R_{\text{TM}}$.

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), κ , 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:

$$\alpha(\lambda)[\text{cm}^{-1}] = \frac{4\pi\kappa}{\lambda} \quad (419)$$

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. Assuming that photon absorption directly creates an electron–hole pair, the optical generation rate of carriers in each element is therefore:

$$G^{\text{opt}}(x, y, z, t) = I(x, y, z)F(t)[1 - e^{-\alpha L}] \quad (420)$$

where $F(t)$ is a time function that is equal to 1 for t in $[t_{\min}, t_{\max}]$ and shows a Gaussian distribution decay outside the interval with the standard deviation σ_t (similar to the time function used in OptBeam, see [Eq. 431, p. 436](#)).

$I(x, y, z)$ is the rate intensity (units of s^{-1}) of the ray in the element. After all of the optical generations 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 $\text{s}^{-1} \cdot \text{cm}^{-3}$. 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 can be activated in different ways using different options:

- Raytracing used in simple optical generation.
- Monte Carlo raytracing.
- Multithreading for raytracing.
- Compact memory model for raytracing.
- Rectangular and circular windows of starting rays.
- Different boundary conditions for raytracing.
- Raytracing used in illumination spectrum (see [Spectral Illumination on page 486](#)).
- Raytracing used in the unified optical interface (see [Raytracing on page 402](#)). Note that, in the unified case, only a reduced set of syntax has been implemented and some syntax has been modified as well.

Optical generation by raytracing is activated by the RayTrace statement in the Physics section. An example of optical generation by raytracing is:

```

Plot{ ...
    OpticalGeneration
    RayTraceIntensity
    RayTrees
}
...
Physics{...
    ComplexRefractiveIndex(
        WavelengthDep(real imag)      # turn on wavelength dependence
        ...
    )
    RayTrace(
        PolarizationVector=(0, 0, 1)
        Print
        Wavelength = 6.0e-5          # [cm]
        WavePower = 0.1              # [W/cm^2]
        WaveDirection = (0, 1, 0)
        RectangularWindow(
            WindowCenter = (0, -100, 0)   # [microns]
            WindowDirection = (0, 1, 0)
            WindowSize = 200             # [micron]
            CellSize = 100
        )
    )
}

```

[Table 176 on page 1144](#) lists the options in the RayTrace statement. Either WaveInt or WavePower can be used to specify the incident beam intensity. In the latter case, either WaveLength or WaveEnergy must be defined to allow for the conversion according to [Eq. 430, p. 435](#).

NOTE The complex refractive index model must be used in conjunction with the raytracer.

WaveDirection denotes the direction of the impinging ray. Raytracing offers two termination conditions:

- Raytracing is terminated if the ray intensity becomes less than n times (MinIntensity specifies n) the original intensity.
- DepthLimit specifies the maximum number of material boundaries that the ray can pass through.

14: Optical Generation

Raytracing

WaveTime specifies the time interval when the ray intensity is constant. WaveTsigma defines the standard deviations of a Gaussian distribution, which describes the temporal decay of the ray intensity outside the time interval.

The starting point of the rays within each beam is defined by a circular or rectangular window (either CircularWindow or RectangularWindow). [Table 165 on page 1136](#) summarizes the keywords that specify a window. For further details and examples, see [Window of Starting Rays on page 414](#). Optical generation by raytracing allows you to define a spatial distribution of the ray intensities within the window (see [Table 165 on page 1136](#) and [Spatial Distribution of Intensity on page 418](#)).

Print creates a .grd file with information about the paths that the rays take. The file name is obtained from the Plot file name by appending _ray.grd. If no Plot file name is specified, the name defaults to Raytrace_ray.grd.

In the Plot section, OpticalGeneration (units of $s^{-1} \cdot cm^{-3}$) gives the optical generation computed by raytracing. RayTraceIntensity (units of W/cm^3) allows you to visualize the optical pattern computed by raytracing within the device. In addition, the keyword RayTrees produces a TDR file containing the entire raytree structure with intensity information that can be visualized and manipulated in Tecplot SV (see [Visualizing Raytracing on page 429](#)).

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 a material 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.

The Monte Carlo raytracing has been implemented in both general raytracing and LED raytracing. The syntax is:

```
Physics {...  
    RayTrace (...  
        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  
    StackSize = 20000000    # increase to e.g. 20MB  
}
```

NOTE Raytracing is a recursive process, so it will obliterate the default (1 MB) stack space if the level of the raytree becomes increasingly deep. This may lead to unexplained segmentation faults or abortion of the program. To resolve this issue, increase the stack space using the keyword `StackSize` in the Math section as shown in the above syntax.

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. If this reduced memory option is activated, plotting or printing of raytrees is not possible since the raytrees are not stored. The command file syntax is:

```
Physics {...
    RayTrace (...  
        CompactMemoryOption
    )
}
```

Window of Starting Rays

In Sentaurus Device raytracing, rays of photon come through some type of window. The window can be either circular or rectangular, although for 2D structures, it makes no difference. The description of a window is designed to be as flexible as possible. The properties of the window that must be defined are:

- The center of the window
- The direction the window faces
- The size of the window
- The number of rays that enter the window

[Table 165 on page 1136](#) summarizes the keywords that specify a window. For more details about the definition of a window, see [Two-dimensional Device and Window Description](#) and [Three-dimensional Device and Window Description on page 416](#).

NOTE If the number of rays is too sparse compared to the element size of the device being simulated, the resulting approximation will be unsatisfactory.

Two-dimensional Device and Window Description

For a 2D device, a window is one dimensional. In this case, the two window types are equivalent.

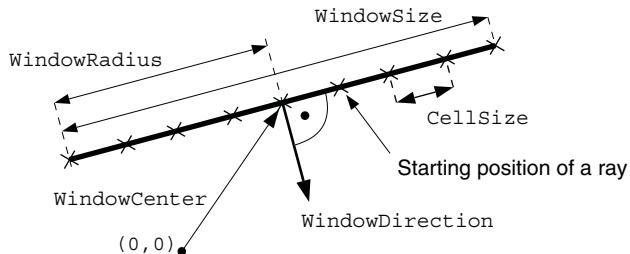


Figure 29 Window for 2D devices

The properties of the 2D device window that must be defined are:

- `WindowCenter` notes the position of the center of the window.
- `WindowDirection` is the normal to the plane of the window, which can be different from the actual wave propagation direction.
- `WindowSize` or `WindowRadius` specifies the size of a 2D device window. `WindowSize` works for both rectangular and circular windows. `WindowRadius` works only for circular windows.
- `CellSize` determines the spacing, which specifies the distance [μm] between ray starting points. Within the window, spatial distribution of the ray is uniform.

Alternatively, `NumberOfRays` can be used to place rays for a two-dimensional device window. If the specified number of rays is N, then N rays are placed uniformly over the window.

The syntax for the 2D device window is:

```
RayTrace(..  
    *** Example for circular window ***  
    CircularWindow(  
        WindowCenter = (10,12,0)  
        WindowDirection = (0,-1,0)  
        WindowRadius = 5  
        CellSize = 0.1  
    )  
  
    *** Example for rectangular window ***  
    RectangularWindow(  
        WindowCenter = (10,12,0)
```

14: Optical Generation

Raytracing

```
    WindowDirection = (0, -1, 0)
    WindowSize = 10
    NumberOfRays = 10
)
)
```

Both examples for circular and rectangular windows result in an identical window.

NOTE By inputting the vector position for 2D, the value of the **z** component must be zero.

Three-dimensional Device and Window Description

For a three-dimensional device, a window is two dimensional. Therefore, it can be either circular or rectangular. This is specified by the keyword `CircularWindow` or `RectangularWindow`, respectively.

The basic properties of the three-dimensional device window are:

- `WindowCenter` is the position of the center of the window.
- `WindowDirection` is the normal to the plane of the window.
- `WindowRadius` (for a circular window) or `WindowHeight/WindowWidth` (for a rectangular window) specifies the size of the 3D window.
- `CellSize` determines the spacing. For a circular window, this is the length of an equilateral triangle that makes up the lattice of ray starting points (see [Figure 30](#)). For a rectangular window, `CellSize` is the length of a side of the square that makes up the lattice of ray starting positions (see [Figure 31 on page 417](#)).
- `WindowOrientation` controls the alignment of the window. For a circular window, one side of the equilateral triangle is aligned to the vector `WindowOrientation`. For a rectangular window, `WindowOrientation` specifies the direction of the side whose length is defined by `WindowHeight` (see [Figure 31](#)).

NOTE If `WindowOrientation` is not specified or the specified orientation is identical to `WindowDirection`, a vector perpendicular to `WindowDirection` is assigned to it.

Figure 30 visualizes the parameters used for a 3D device with a circular window.

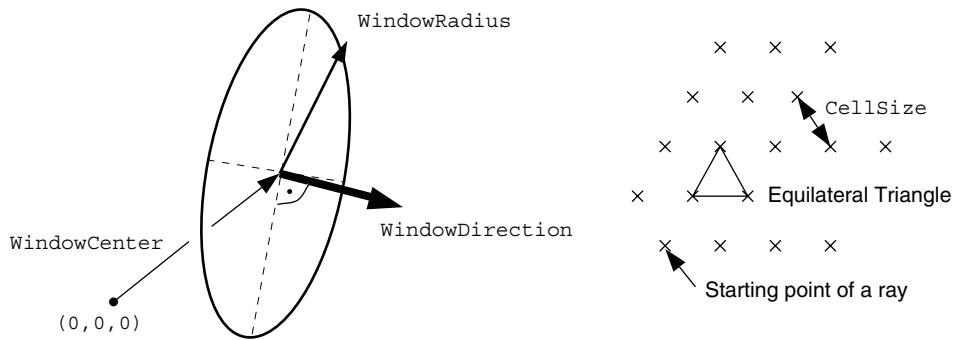


Figure 30 Circular window for 3D device and its lattice of rays

An example of a circular window is:

```
RayTrace(...  
CircularWindow(  
    WindowCenter = (10,12,0)  
    WindowDirection = (0,-1,0)  
    CellSize = 0.1  
    WindowRadius = 5  
    WindowOrientation = (0,0,1)  
)  
)
```

Figure 31 visualizes the parameters used for a 3D device with a rectangular window.

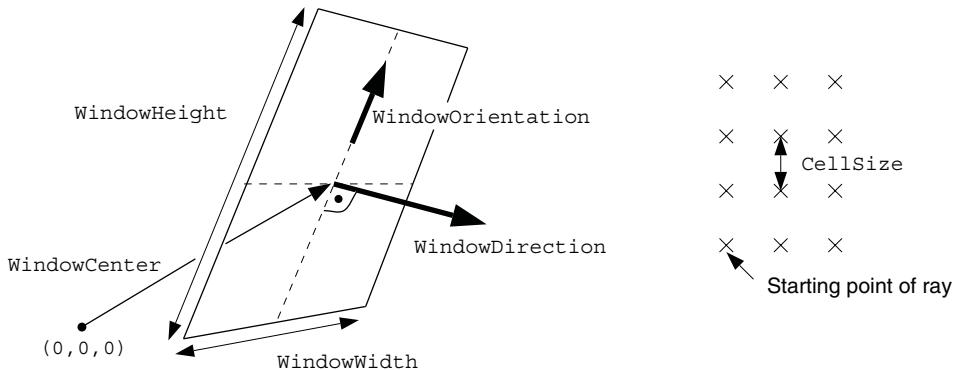


Figure 31 Rectangular window for 3D device and its lattice of rays

An example of a rectangular window is:

```
RayTrace(...  
RectangularWindow(  
    WindowCenter = (10,12,0)  
    WindowDirection = (0,-1,0)
```

14: Optical Generation

Raytracing

```
    CellSize = 0.1  
    WindowHeight = 5  
    WindowWidth = 10  
    WindowOrientation = (0,0,1)  
  )  
)
```

Spatial Distribution of Intensity

In Sentaurus Device, the intensity of rays within a given window can vary by using the following keywords in the CircularWindow or RectangularWindow statement group (see [Table 165 on page 1136](#)):

- `intvalue1 | intvalue2` specify the values for the spatially varying intensity spline function. For a circular window, `intvalue1` specifies the intensity distribution along the ‘radius’ direction. For a rectangular window, `intvalue1 | intvalue2` specify the intensity distribution in the ‘height’ and ‘width’ direction.
- `intposition1 | intposition2` indicate where each point in `intvalue1 | intvalue2` occurs on the window, in the absolute length of the window.

Alternatively, `intPositionNormalized1 | intPositionNormalized2` specify where each point in `intvalue1 | intvalue2` occurs on the window, in the normalized length of the window.

If neither `intPositionNormalized1 | intPositionNormalized2` nor `intposition1 | intposition2` is specified, a uniform spacing of data points is assumed.

The `intvalue1 | intvalue2` statement defines a linear spline function **f(d)** and **g(e)**, where **d** and **e** are the distances from the center of the window.

[Figure 32](#) visualizes the spline function for `intvalue1=[1,1,0.5,0.3,0.3,0]` and `intvalue2=[1,0.5,0.5,0.25,0]` in the case of a rectangular and a circular window.

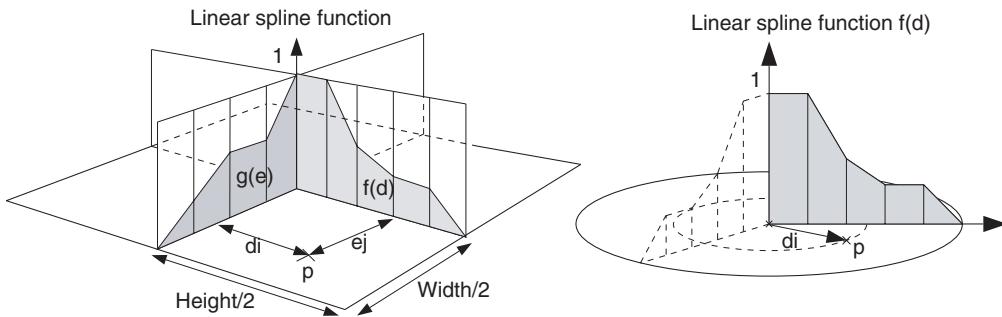


Figure 32 Linear spline for rectangular window (*left*) and circular window (*right*)

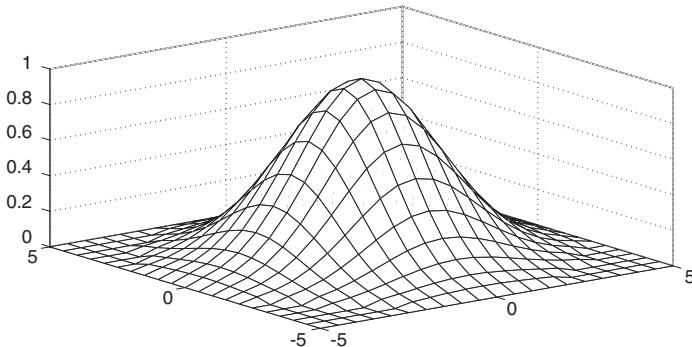
For a given ray r , starting in the window position p , with intensity \mathbf{I} , specified by the waveint command, the starting intensity is modified according to:

- In the presence of intvalue1: $I_{i,internal} = f(\mathbf{d}_i) * \mathbf{I}$
- In the presence of intvalue1 and intvalue2 (for a rectangular window):
 $I_{i,internal} = f(\mathbf{d}_i) * g(\mathbf{e}_j) * \mathbf{I}$

An example of a circular window resulting in a Gaussian intensity profile is:

```
RayTrace(...  
CircularWindow(  
    WindowCenter = (10,12,0)  
    WindowDirection = (0,-1,0)  
    CellSize = 0.1  
    WindowRadius = 5  
    intvalue1 = [ 1.0000  0.9610  0.8529  0.6990  
                0.5291  0.3698  0.2387  0.1423  
                0.0784  0.0398  0.0187  0.0000 ]  
)  
)
```

For a 3D device, the above code creates a window with an intensity profile as in [Figure 33](#).



[Figure 33](#) Intensity in a 3D device circular window with intval1

NOTE No rays are defined outside the window, in which case, [Figure 33](#) would be a circle of radius 5.

An example of a rectangular window with only intvalue1 specified is:

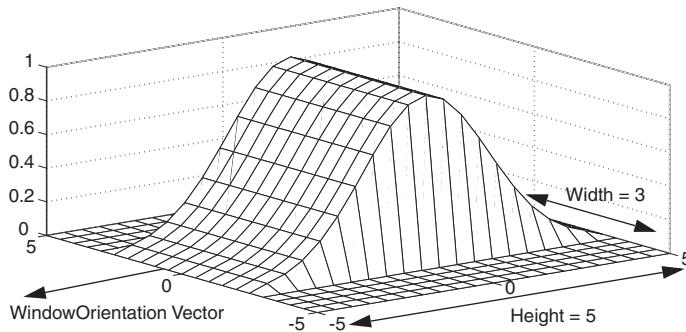
```
RayTrace(...  
RectangularWindow(  
    WindowCenter = (10,12,0)  
    WindowDirection = (0,-1,0)  
    CellSize = 0.1  
    WindowHeight = 5  
    WindowWidth = 3  
    WindowOrientation = (0,0,1)
```

14: Optical Generation

Raytracing

```
intvalue1 = [ 1.0000  0.9610  0.8529  0.6990  
            0.5291  0.3698  0.2387  0.1423  
            0.0784  0.0398  0.0187  0.0000 ]  
        )  
)
```

For a 3D device, the above input will result in the intensity profile as shown in [Figure 34](#).



[Figure 34](#) Three-dimensional rectangular window with only intval1 specified

An example of a 3D device with a rectangular window, where intvalue1 and intvalue2 specify a Gaussian intensity distribution, is:

```
RayTrace(...  
    RectangularWindow(  
        WindowCenter = (10,12,0)  
        WindowDirection = (0,-1,0)  
        CellSize = 0.1  
        WindowHeight = 5  
        WindowWidth = 3  
        WindowOrientation = (0,0,1)  
        intvalue1 = [ 1.0000  0.9610  0.8529  0.6990  
                    0.5291  0.3698  0.2387  0.1423  
                    0.0784  0.0398  0.0187  0.0000 ]  
        intvalue2 = [ 1.0000  0.9610  0.8529  0.6990  
                    0.5291  0.3698  0.2387  0.1423  
                    0.0784  0.0398  0.0187  0.0000 ]  
    )  
)
```

Identical to values in the intvalue1 string of numbers, the resulting spline function from intvalue2 is similar in shape to that resulting from intvalue1, but thinner by a ratio of 3:5, the ratio of the window sizes. The following intensity pattern will arise for the 3D device with rectangular windows, where intvalue1 and intvalue2 are specified (see [Figure 35](#) on page 421).

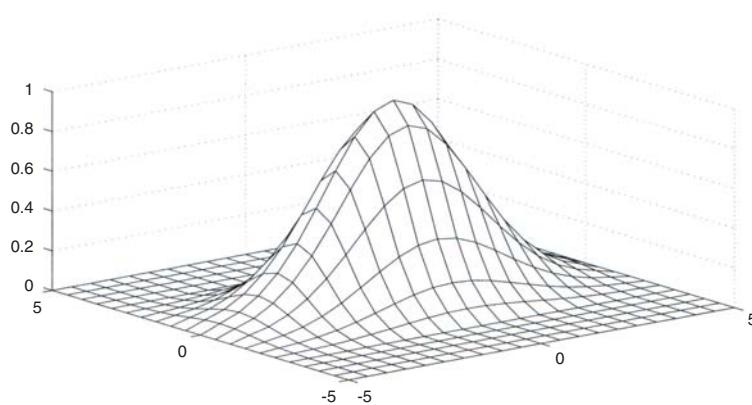


Figure 35 Three-dimensional device rectangular window intensity pattern

Use the keywords `intpositionnormalized1` and `intpositionnormalized2` to specify where each point in `intvalue1` and `intvalue2` occurs on the window. Thereby, values for `intpositionnormalized1` and `intpositionnormalized2` indicate the normalized length of the window. Rather than using normalized lengths to the window size, absolute lengths in the intensity position field can be specified by using `intposition1` and `intposition2`, respectively.

An example of a sharper drop from value 1 to 0.5 at $d = 2 \mu\text{m}$ is shown in [Figure 36 on page 422](#). Therefore, `intpositionnormalized1` or `intposition1` can be used:

```
RayTrace(...  
    CircularWindow(  
        WindowCenter = (10,12,0)  
        WindowDirection = (0,-1,0)  
        CellSize = 0.1  
        WindowRadius = 5  
        intvalue1 = [1, 1, 0.5, 0.3, 0.3, 0]  
  
        *** Use normalized lengths of the window ***  
        intpositionnormalized1 = [0, 0.39, 0.4, 0.6, 0.8, 1]  
  
        *** Alternatively, use absolute lengths of the window ***  
        intposition1 = [0, 1.99, 2, 3, 4, 5]  
    )  
)
```

14: Optical Generation

Raytracing

The resulting spline function appears as shown in [Figure 36](#).

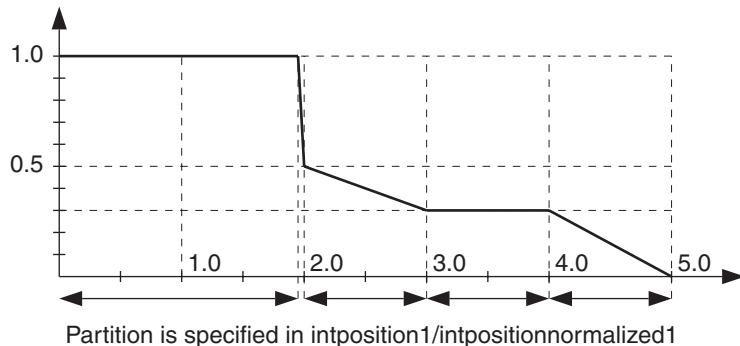


Figure 36 Spline function with nonuniform spacing

If there is a mismatch in length, Sentaurus Device assumes that only the vector `intvalue1` or `intvalue2` is given and forms a uniform length partition. In the case of a non-monotone increasing position field, either in `intposition1|intposition2` or `intpositionnormalized1|intpositionnormalized2`, the non-increasing portion is ignored.

Boundary Condition for Raytracing

Special and spatially arbitrary boundary conditions can be specified in raytracing. The boundaries are drawn as contacts (see [Electrode Section on page 50](#) and [Thermode Section on page 53](#)) and labeled accordingly using Sentaurus Structure Editor. These contacts can be specifically constructed for setting a raytrace boundary condition, or they can coincide with an electrode or a thermode. The contact is discretized into edges (2D) or faces (3D) after meshing.

Each edge/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/face, the higher ranked one is chosen:

1. Constant reflectivity/transmittivity contact
2. Raytrace PMI contact
3. Multilayer AR coating contact
4. Photon recycling contact

NOTE To define clearly special raytrace boundary conditions, each line/surface contact defined should have a distinct name. This means that two parallel contacts should have different contact names, and intersecting contacts should also have different contact names.

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
    }
    ...
}
```

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 37](#) and [Figure 38 on page 424](#). In the 2D case, a star boundary is drawn within a device; in the 3D case, a rectangular boundary is drawn within the device.

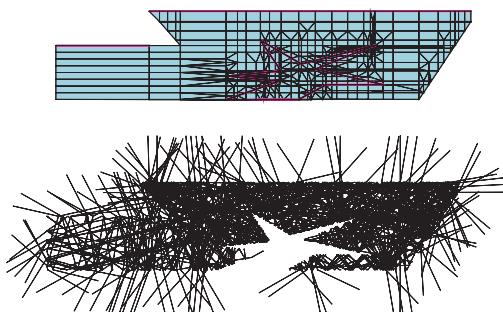


Figure 37 Applying the reflecting boundary condition in a 2D LED simulation; the boundary is drawn as a star inside the device

14: Optical Generation

Raytracing

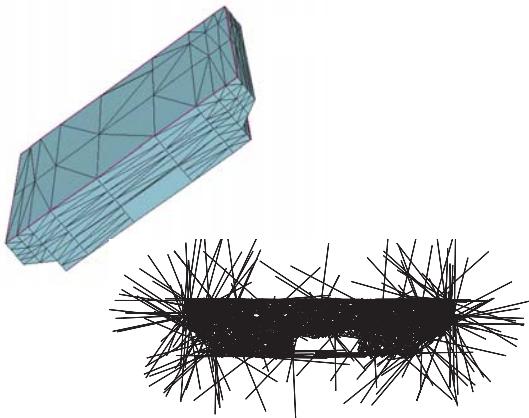


Figure 38 Applying the reflecting boundary condition in a 3D LED simulation; the boundary is drawn as a hollow rectangular waveguide inside the device

Raytrace PMI Boundary Condition

A special raytrace PMI contact 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 [Special Contact PMI for Raytracing on page 1033](#).

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 special contact and include the following syntax in the RayTraceBC section of the command file:

```
RayTraceBC { ...
    { Name = "pmi_contact"
        PMIModel = "pmi_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.

Thin-Layer-Stack Boundary Condition

A thin-layer-stack boundary condition can be used to model interference effects in raytracing. The modeling of antireflection coatings used in solar cells is a typical example of the use of this 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 438](#)), 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 39 on page 426](#), shows the use of this boundary condition:

```
RayTraceBC {
    { Name="rayContact1" reflectivity=1.0 }
    { Name="rayContact2"
        ReferenceMaterial = "Gas"
        LayerStructure {
            70e-3 "Nitride";# micron
            6e-3 "Oxide" # micron
        }
    }
    { Name="rayContact3"
        ReferenceRegion = "region_on_top"
        LayerStructure {
            10e-3 "Oxide"; # micron
        }
    }
}
```

The first line in the RayTraceBC section below shows the definition of a constant reflectivity as a boundary condition (see [Constant Reflectivity and Transmittivity Boundary Condition on page 423](#)). This option is usually chosen for the boundary of the simulation domain.

The other option is to define a multilayer structure, for which the corresponding contact in the grid file can be seen as a placeholder. For each layer, the corresponding thickness ([μm]) and material name must be specified. For the calculation of the transmittance and reflectance, the transfer matrix method reads the complex refractive index from the TableODB in the respective parameter file. An additional parameter file or a 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.

14: Optical Generation

Raytracing

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 that it is only possible to 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.

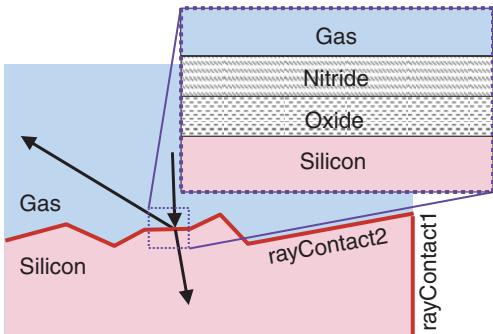


Figure 39 Illustration of thin-layer-stack boundary condition for simulation of an antireflection coated solar cell

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 [Virtual Regions in Raytracer on page 427](#) to ignore these thin layers in the raytracing process. The required syntax is:

```
RayTraceBC { ...
  { Name="TMMcontact"
    ReferenceMaterial = "Gas"
    LayerStructure {...}
    MapOptGenToRegions {"thinlayer1" "thinlayer2" ...}
    QuantumEfficiency = float
  }
}
```

```
Physics {...  
    RayTrace (...  
        VirtualRegions("toppml" "nitride" "oxide" ...)  
    )  
}
```

A few comments about the syntax:

- QuantumEfficiency denotes the fraction of absorbed photons in the TMM contact that will be converted to optical generation.
- The region names in the MapOptGenToRegion section refer to regions in the electrical device grid.
- The region names in the VirtualRegions section refer to regions in the optical device grid.

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 {...  
    RayTrace (...  
        VirtualRegions("nitride" "oxide" "layer555" ...)  
    )  
}
```

Additional Options for Raytracing

Several options enable better control of raytracing in Sentaurus Device:

- Omitting reflected rays when performing raytracing.
- Omitting weaker rays when performing raytracing.
- The number of ray paths can be reduced by including the Skip(<integer>) keyword in the Print statement. For example, Print(Skip(5)) will only print every other fifth ray path.
- The status of building the raytree can be checked by using the keyword ProgressMarkers, which shows the incremental percentage of the completion of the raytree-building process.

14: Optical Generation

Raytracing

The syntax for these options is:

```
Physics { ...
    RayTrace (
        OmitReflectedRays
        OmitWeakerRays
        Print (Skip(<integer>))
        ProgressMarkers = <integer> # between 1 and 100
    )
}
```

Optical Generation Scaling for Raytracing

The optical generation profile can be scaled by a global factor in raytracing. This applies to both general raytracing and LED raytracing. The syntax is:

```
Physics {...
    RayTrace (...  
        OptGenScaling = 100.0
    )
}
```

Redistributing Power of Stopped Rays

When the 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. The total power of the rays is:

$$P_{\text{Total}} = P_{\text{abs}} + P_{\text{escape}} + P_{\text{stopped}} \quad (421)$$

where P_{abs} is the absorbed power, P_{escape} is the power of the escaped rays (out of the device), and P_{stopped} is the power of the rays terminated by the DepthLimit or MinIntensity condition. Rearranging the equation, the following expression is obtained:

$$P_{\text{Total}} = \frac{1}{\left(1 - \frac{P_{\text{stopped}}}{P_{\text{Total}}}\right)} (P_{\text{abs}} + P_{\text{escape}}) \quad (422)$$

Therefore, by multiplying a redistribution factor to P_{abs} and P_{escape} , the leftover power in the stopped rays can be accounted for. The syntax is:

```
Physics {...  
    RayTrace (...  
        RedistributeStoppedRays  
    )  
}
```

In addition, the syntax applies to both general raytracing and LED raytracing.

NOTE This feature does not work with Monte Carlo raytracing due to the fundamental assumption of the Monte Carlo method.

Visualizing Raytracing

The `Print` option in the `Raytrace` section creates a `.grd` file with information about the paths that the rays take (see [Using the Raytracer on page 410](#)).

Alternatively, 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.

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 W/cm^2) by the rectangular or circular window area where the starting rays have been defined.
- 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.

14: Optical Generation

Raytracing

- 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.

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 may lead to the unnecessary deterioration of simulation performance. In such 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 40](#)).

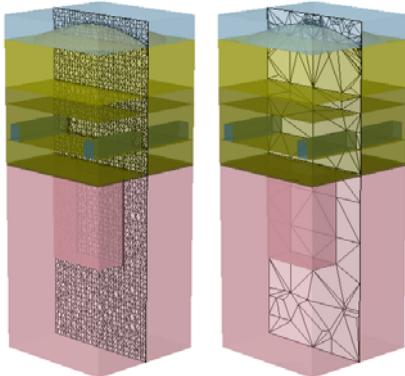


Figure 40 Comparison of two grids used in a 3D dual-grid CMOS image sensor simulation; the figures show cuts through the device center; the electronic equations are solved on a fine grid (*left*) and a coarse grid is used for raytracing (*right*)

The basic syntax for setting up a raytracing dual-grid simulation is:

```
File {  
    Output  = "output"  
    Current = "current"  
}  
Plot {  
    RayTraceIntensity  
    OpticalGeneration  
}  
  
# Specify grid and material parameter file names for raytracing:  
OpticalDevice OptGrid {  
    File {
```

```

        Grid      = "raytrace.grd"
        Doping    = "raytrace.dat"
        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.grd"
        Doping    = "in_elec_pof.dat"
        Parameter = "CIS.par"
        Current   = "current"
        Plot      = "plot"
    }
    Physics {
        AreaFactor = 1
        ComplexRefractiveIndex(... 
            WavelengthDep(real imag)
        )
        RayTrace (
            PolarizationVector=(1,0,0)
            WavePower  = 1e-4
            WaveLength = 0.5e-4
            WaveDirection = (0,0,-1)
            DepthLimit = 6
            MinIntensity = 1e-3
            RectangularWindow (
                WindowCenter = (0,0,5.0)
                WindowOrientation = (1,0,0)
                WindowHeight = 1.5
                WindowWidth = 1.5
                Cellsize = 0.085
            )
        )
    }
}

# Specify system connectivity:
System {
    OptGrid opt ()

```

14: Optical Generation

Raytracing

```
CIS d1 (pd=vdd sub=0) { Physics{ OptSolver = "opt" } }
Vsource_pset drive(vdd 0){ dc = 0.0 }
}

Solve { Poisson }
```

Old Raytracer

NOTE The use of the old raytracer is discouraged as it will be removed in a future release. The description remains for completeness and reference only.

This section describes the theory of the old raytracer, which uses the concept of polarization ellipse. To activate the old raytracer, use the syntax:

```
Physics {...  
    RayTrace(...  
        UseOldRayTracer  
    )  
}
```

When a ray passes through a material boundary, it splits into two rays. The angles involved are governed by Snell's law (see [Figure 41](#)):

$$n_1 \sin(\theta_i) = n_2 \sin(\theta_t) \quad (423)$$

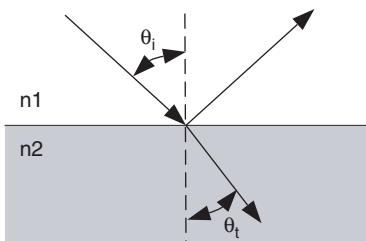


Figure 41 Snell's law

The following formulas describe the intensity of the refracted electromagnetic wave on a plane surface between two media:

$$T_{\text{perp}} = \frac{\sin(2\theta_i)\sin(2\theta_t)}{\sin^2(\theta_i + \theta_t)} \quad (424)$$

$$T_{\text{par}} = \frac{\sin(2\theta_i)\sin(2\theta_t)}{\sin^2(\theta_i + \theta_t)\cos^2(\theta_i - \theta_t)} \quad (425)$$

The above formulas decompose the plane wave into a polarization component that is perpendicular to the plane of incidence (spanned by the ray propagation vector and the surface normal) and a component parallel to the plane of incidence. Reflection coefficients can be computed by subtracting the transmission coefficients from 1:

$$1 - T_{\text{par}} = R_{\text{par}} \quad (426)$$

$$1 - T_{\text{perp}} = R_{\text{perp}} \quad (427)$$

The possibility of absorption of a photon on the interface is disregarded.

An incident ray on metal is assumed to be reflected completely.

Polarization information regarding a ray can be stored as pairs of points that form an ellipse. Thereby, a point on the ellipse represents a component in the photon beam that is polarized in that direction. The distance of the point from the origin is the intensity of that component. The total intensity of a ray is an integral of the distance of each point from the center of the ellipse.

For example, let the initial polarization ellipse have minor and major axes of $2a$ and $2b$, respectively (see [Figure 42](#)).

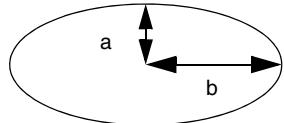


Figure 42 Polarization ellipse

$$I = I_0 \int_0^{2\pi} \sqrt{a^2 \sin^2 \theta + b^2 \cos^2 \theta} d\theta \quad (428)$$

The intensity is described by [Eq. 428](#), where I_0 is the original ray intensity. The above integral does not have a closed form solution. The second formula of Ramanujan is used, which approximates the above integral by:

$$I = I_0 \pi(a+b) \left[1 + \frac{3h}{10 + \sqrt{4-3h}} \right] \quad (429)$$

$$\text{where } h = \left(\frac{a-b}{a+b} \right)^2.$$

Polarization of Old Raytracer

Initial polarization of a ray can be specified by the `Polarization` statement, which must have the following format:

```
RayTrace(...  
  Polarization(  
    axis1 = a  
    axis2 = b  
    axis1dir = v  
  )  
)
```

where `a` and `b` are real numbers, and `v` is a vector in three-dimensional space. The ratio between `a` and `b` is important, not their absolute values. The vector `v` specified in the input does not need to be orthogonal to `WindowDirection`; it only needs to be non-identical to it. To determine the polarization of the ray, vector `v` is projected to the plane orthogonal to the ray direction (see [Figure 43](#)). `C` is a constant that ensures the intensity represented by the ellipse has a correct value.

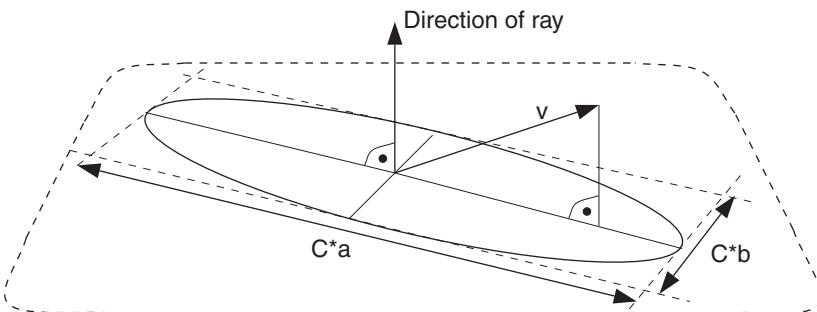


Figure 43 Polarization of a ray

The keywords for `Polarization` are listed in [Table 165 on page 1136](#).

Optical Beam Absorption

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 44 illustrates one optical beam and its optical intensity distribution in a 3D device.

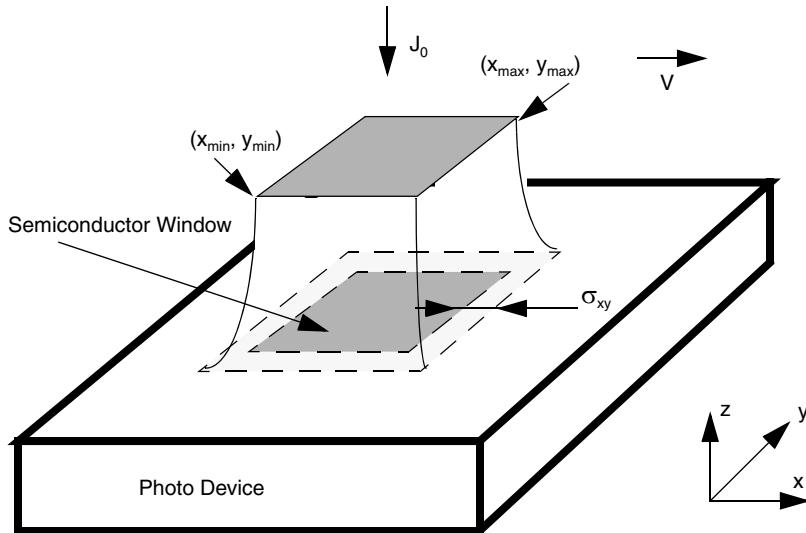


Figure 44 Intensity distribution of an optical beam

J_0 denotes the optical beam intensity (number of photons that cross an area of 1 cm^2 per 1 s) at the center of the semiconductor window. (x_{\min}, y_{\min}) and (x_{\max}, y_{\max}) define the semiconductor window, σ_{xy} is the standard deviation of the spatial Gaussian distribution that describes the decay of the optical beam intensity outside the semiconductor window, \vec{V} is the velocity of the semiconductor window. The space shape F_{xyv} of the incident beam intensity is defined by the semiconductor window size, its spatial Gaussian decay, and its velocity. Thereby, F_{xyv} is equal to 1 inside the semiconductor window and decreases to zero according to Gaussian distribution outside the device.

The following equations describe useful relations for the photogeneration problem:

$$E_{\text{ph}} = \frac{hc}{\lambda} \quad (430)$$

$$J_0 = \frac{P_0}{E_{\text{ph}}}$$

where P_0 is the incident wave power per area [W/cm^2], λ is the wavelength [cm], h is the Planck constant [J s], c is the speed of light in a vacuum [cm/s], and E_{ph} is the photon energy that is approximately equal to $\frac{1.24}{\lambda[\mu\text{m}]}$ in eV.

14: Optical Generation

Optical Beam Absorption

The default model computes the optical generation rate along the z-axis according to:

$$G^{\text{opt}}(z, t) = J_0 F_t(t) F_{xyv} \cdot \alpha(\lambda, z) \cdot \exp(-\alpha|z - z_0|) \quad (431)$$

where:

- t is the time.
- $F_t(t)$ is the beam time behavior function that is equal to 1 for t in $[t_{\min}, t_{\max}]$ and shows a Gaussian distribution decay outside the interval with the standard deviation σ_t .
- z_0 is the coordinate of the semiconductor surface.
- $\alpha(\lambda, z)$ is the nonuniform absorption coefficient along the z-axis.

A more advanced formulation reads:

$$G^{\text{opt}}(z, t) = J_0 F_t(t) F_{xyv} \cdot \alpha(\lambda, z) \cdot \exp\left(-\int_{z_0}^z \alpha(\lambda, z') dz'\right) \quad (432)$$

This model is suggested for devices with varying absorption coefficient along the z-axis.

Using Optical Beam Absorption

Optical generation by simple optical beam absorption is activated by using the OptBeam statement in the Physics section. You can specify any number of beams, for example, several beams with different wavelengths. In addition, if the OptBeam statement is specified in the region or material Physics section, the optical generation is specified for this region or material.

An example of optical beam absorption is:

```
Plot {  
    ...  
    OptBeam  
}  
...  
Math {  
    ...  
    RecBoxInteger (1e-3 10 1000)  
}  
...  
Physics {  
    ...  
    OptBeam (  
        WaveLength = 7.00e-05 # [cm]  
        WavePower = 0.1 # [W/cm^2]  
        SemAbs (model=...)
```

```

Semsurf = 0.0
SemWind = (0.0 0.02) )
( WaveLength = ...
...
SemWindow = ... )
...
)
}

```

[Table 172 on page 1141](#) lists the keywords to define a beam in the OptBeam statement. Either WaveInt or WavePower can be used to specify the incident beam intensity. In the latter case, either WaveLength or WaveEnergy must be defined to allow for conversion according to [Eq. 430](#). Different types of absorption model can be activated using SemAbs. All options that are described in [Absorption Models on page 473](#) are valid for the optical beam absorption model.

SemSurf specifies the coordinate z_0 that defines the semiconductor surface. If z_0 is located inside the device, the maximum of the beam intensity ($J_0(\alpha(\lambda, z_0))$) is at this coordinate, and two beams with opposite directions are applied starting from this point.

SemWindow specifies the coordinates of the semiconductor window. WaveTime specifies a time interval when the incident beam intensity is constant.

WaveXYSigma and WaveTsigma both define the standard deviation of a Gaussian distribution. The first constant describes the spatial decay of the incident beam intensity outside the semiconductor window. The second constant describes the temporal decay of the incident beam intensity outside the time interval. The optical beam absorption model supports the movement of the semiconductor window. Thereby, SemVelocity describes the velocity of the window perpendicular to the direction of the incident beam.

The keyword RecBoxIntegr in the Math section activates the optical generation computation according to [Eq. 432](#). For varying absorption coefficients, RecBoxIntegr is recommended. BeamGeneration in the Plot section allows for the visualization of the optical generation by optical beam absorption.

If the beam intensity changes rapidly in space, the accuracy of the beam intensity computation is low for coarse meshes unless special precaution is taken. Such problems can be eliminated by increasing the accuracy of the beam distribution integration over the control volume associated with each mesh vertex. Such integrations are performed by inserting small rectangular boxes inside the control volume and by executing analytic integration inside the small boxes.

To activate this additional procedure, the keyword RecBoxIntegr is specified in the Math section with three additional parameters to control the accuracy of the procedure:

```
RecBoxIntegr(<Epsilon> <MaxNumberOfLevels> <MaxNumberOfBoxes>)
```

A specification without any parameters defaults to `RecBoxIntegr(1e-2 10 1000)`. The first value is the maximum relative deviation of the volume covered by rectangles from the box volume. The second value is the maximum number of refinement levels, and the third value is the maximum number of rectangles used to approximate each box. [Figure 45](#) illustrates these rectangular boxes.

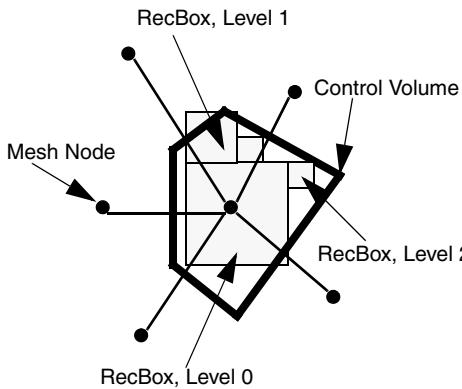


Figure 45 Example of rectangular boxes without parameter defaults

Transfer Matrix Method

Sentaurus Device can calculate the propagation of plane waves through layered media by using a transfer matrix approach. An extension for inverted pyramid structures as they are used for high-efficiency solar cells allows for the modeling of light propagation in such structures by appropriately transforming them into planar-layered media with similar optical properties.

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 linear. In this case, the amplitudes of forward and backward running waves A_j^\pm and B_j^\pm in each layer in [Figure 46 on page 439](#) 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 Θ_j is the complex counterpart of the angle of refraction ($n_0 \cdot \sin\Theta_0 = n_j \cdot \sin\Theta_j$).

Real and complex parts of the complex refractive index $\tilde{n} = n + ik$ can be defined using the keywords `Refract` and `Absorption` in the `Layer` section of the input file or can be read from

the TableODB section in the parameter file (see [Table-based Optical Properties of Materials in Parameter File on page 475](#)).

The transfer matrix of the interface between layers j and $j + 1$ is defined by:

$$T_{j,j+1} = \frac{1}{2Z_j} \cdot \begin{bmatrix} Z_j + Z_{j+1} & Z_j - Z_{j+1} \\ Z_j - Z_{j+1} & Z_j + Z_{j+1} \end{bmatrix} \quad (433)$$

The propagation of the plane waves through layer j can be described by the transfer matrix:

$$T_j(d_j) = \begin{bmatrix} \exp\left(2\pi i n_j \cos \Theta_j \frac{d_j}{\lambda}\right) & 0 \\ 0 & \exp\left(-2\pi i n_j \cos \frac{\Theta_j d_j}{\lambda}\right) \end{bmatrix} \quad (434)$$

with the thickness d_j of layer j and the wavelength λ of the incident light. The transfer matrices connect the amplitudes of [Figure 46](#) as follows:

$$\begin{aligned} \begin{pmatrix} B_j^+ \\ A_j^+ \end{pmatrix} &= T_{j,j+1} \cdot \begin{pmatrix} A_{j+1}^- \\ B_{j+1}^- \end{pmatrix} \\ \begin{pmatrix} A_j^- \\ B_j^- \end{pmatrix} &= T_j(d_j) \cdot \begin{pmatrix} B_j^+ \\ A_j^+ \end{pmatrix} \end{aligned} \quad (435)$$

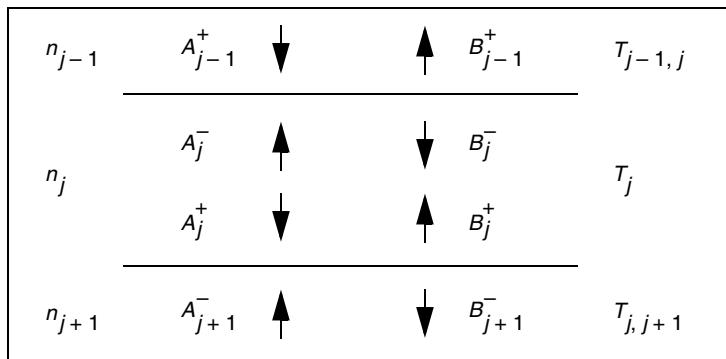


Figure 46 Wave amplitudes in a layered medium and transfer matrices connecting them

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:

$$I_{T(\text{TE, TM})}(d) = \frac{\Re(Z_j)}{\Re(Z_0)} \cdot \left\| T_j(d) \cdot \begin{pmatrix} A_j \\ B_j \end{pmatrix} \right\|^2 \quad (436)$$

with the proper wave impedances. If δ is the angle between the vector of the electric field and the plane of incidence, the intensities have to be added according to:

$$I(d) = I_{\text{TM}}(d) + I_{\text{TE}}(d) \quad (437)$$

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^{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 α 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:

$$G_0^{\text{opt}} = \alpha \eta \frac{I(d)}{\hbar\omega} \quad (438)$$

where the absorption coefficient α is given by the imaginary part of $4\pi Z_{\text{Si}}/\lambda$. The quantum yield η is defined as the number of carrier pairs generated by one photon. Up to photon energies of 3 eV, η equals one. With increasing energy, it increases linearly in first approximation to a value of 3 at 6 eV [1][2]. The additional pairs are created by impact ionization.

The quantum yield is given by:

$$\eta = \begin{cases} 1 + 33.5(0.45\mu\text{m} - \lambda[\mu\text{m}])^2 & \text{for } \lambda < 0.450\mu\text{m} \\ 1 & \text{else} \end{cases} \quad (439)$$

unless it is defined using the keyword `QuantumYield` in the `OpticalGeneration` section of the input file.

Under the influence of ultraviolet radiation, the generated electron-hole pairs possess kinetic energies of at least 1 eV, that is, 40 times the thermal energy $k_B T$. As drift-diffusion equations handle only carriers in thermal equilibrium, the generated electron-hole pairs must not be taken into account in the simulation until they are thermalized. During cooling, they diffuse from the location of generation. Therefore, the generation rate G_0^{opt} must be spread out with a suitable weight function.

The spreading is implemented with two weight functions, the stepwise constant function:

$$c(x) = \begin{cases} \frac{1}{2\lambda_{\text{sp}}} & -\lambda_{\text{sp}} \leq x \leq \lambda_{\text{sp}} \\ 0 & \text{otherwise} \end{cases} \quad (440)$$

and the Gaussian function:

$$g(x) = \frac{1}{\lambda_{\text{sp}}\sqrt{\pi}} \cdot \exp\left[-\left(\frac{x}{\lambda_{\text{sp}}}\right)^2\right] \quad (441)$$

Both functions are normalized:

$$\int_{-\infty}^{+\infty} c(x) dx = \int_{-\infty}^{+\infty} g(x) dx = 1 \quad (442)$$

The modified optical generation G^{opt} is the convolution of G_0^{opt} from Eq. 438 and a weight function, where those carriers that would diffuse out of the silicon layer are mirrored back with a loss factor β :

$$G^{\text{opt}}(d) = \left(\int_0^{\infty} G_0^{\text{opt}}(x) w(d(-x')) dx' \right) + (1 - \beta) \int_0^{\infty} G_0^{\text{opt}}(x) w(d + x')(dx') \quad (443)$$

Here, d is the distance from the upper boundary of the silicon layer and w is one of the weight functions. The characteristic length λ_{sp} can be calculated with a suggested random walk model [3]:

$$\lambda_{\text{sp}} = \sqrt{\frac{2}{3} N_{\text{ph}}} \lambda_{\text{ph}} \quad (444)$$

where $\lambda_{\text{ph}} = 5.5 \text{ nm}$ is the average mean free path for phonon scattering and N_{ph} is the number of phonons generated during thermalization.

Assuming that optical phonon scattering and impact ionization are the determining mechanisms in the thermalization process, the number of phonons N_{ph} generated during this process is given by:

$$N_{\text{ph}} = \frac{1}{2} \frac{\hbar\omega - E_{\text{gap}} - (\eta - 1)\langle E_{\text{imp}} \rangle}{\langle E_{\text{ph}} \rangle} \quad (445)$$

where E_{gap} is the band gap. $\langle E_{\text{imp}} \rangle = 1.5 \text{ eV}$ and $\langle E_{\text{ph}} \rangle = 0.054 \text{ eV}$ [4] are the average impact ionization and phonon energies, respectively.

The weight functions are defined by assigning the keywords None, Constant, or Gaussian to the keyword Spreading in the OpticalGeneration section of the input file. The characteristic spreading length is calculated according to Eq. 444 and Eq. 445 unless it is defined using the keyword Lambda in the OpticalGeneration section. The term β is defined with the keyword Loss in the OpticalGeneration section.

Using Transfer Matrix Method

NOTE The transfer matrix method is also supported by the [Unified Interface for Optical Generation Computation on page 393](#). Refer to [Solving the Optical Problem on page 400](#) for a description of its use in this context.

Two sections inside the Physics section of the command file must be specified for the calculation of the optical generation using the transfer matrix method. The OpticalGeneration section activates the computation of the optical generation along with optional parameters such as the quantum yield. The Optics section determines the underlying optical solver together with solver-specific parameters as shown in the following example:

```
OpticalGeneration (
    ...
    QuantumYield = 1
)
Optics (
    TMM (

```

```

NodesPerWavelength = 20
Excitation (
    Wavelength = 0.3 # [um]
    WavePower = 1000 # [W/m^2]
    Polarization = 0.5
    Theta = 20 # [deg]
)
Stripe (
    Left = 0 # [um]
    Right = 10 # [um]
)
Stripe (
    ...
)
)

```

The properties of the incident light such as Wavelength, WavePower, Polarization, and angle of incidence Theta are specified in the Excitation section of the TMM section. Every Stripe in the TMM section specifies a Left and Right marker between which the corresponding one-dimensional complex refractive index profile will be extracted automatically from the grid file. Based on this profile, the polarization-dependent optical intensity is calculated and interpolated onto the grid within the boundaries defined in the Stripe definition. The optical generation profile is then calculated from the stripe-specific intensities. For overlapping stripes, the intensity of a previously defined stripe is overwritten by the intensity of the current stripe and so on.

By default, the surrounding media at the top and bottom of the extracted layer structure are assumed to have the material properties of vacuum. However, the default can be overwritten by specifying a separate `Medium` section in the `Stripe` section for the top and bottom of the layer structure if necessary. In the `Medium` section, a `Location` must be specified and a `Material`, or, alternatively, the values of `RefractiveIndex` and `ExtinctionCoefficient`:

```
Stripe (
    Left = 0 #[um]
    Right = 10 #[um]
    Medium (
        Location = top
        Material = "Silicon"
    )
    Medium (
        Location = bottom
        RefractiveIndex = 1.4
        ExtinctionCoefficient = 1e-3
    )
)
```

The keywords of `OpticalGeneration` are listed in [Table 173 on page 1142](#). `QuantumYield` is used to redefine the quantum yield given by [Eq. 439](#). The keywords of `TMM` are summarized in [Table 180 on page 1147](#).

If the optical generation profile is the sole quantity of interest, it is sufficient to specify the keyword `Optics` exclusively in the `Solve` section.

By default, the optical generation rate is calculated for every semiconductor region. However, it is possible to 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" ) { -OpticalGeneration }
Physics ( material="InP" ) { -OpticalGeneration }
```

NOTE The values for the complex refractive index are taken from the `TableODB` section in the parameter file (see [Table-based Optical Properties of Materials in Parameter File on page 475](#)).

To visualize optical intensity and generation, the keywords `OpticalIntensity` and `OpticalGeneration` must be added to the `Plot` section:

```
Plot {
  ...
  OpticalIntensity
  OpticalGeneration
}
```

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:

$$I = \text{Re}(A + B)^2 + \text{Im}(A + B)^2 \quad (446)$$

whereas using the syntax:

```
TMM (
  ...
  IntensityPattern = Envelope
)
```

computes the optical intensity I using the following formula to prevent oscillations on the wavelength scale that may not be possible to resolve on the mixed-element simulation grid:

$$I = \text{Re}(A)^2 + \text{Im}(A)^2 + \text{Re}(B)^2 + \text{Im}(B)^2 \quad (447)$$

where A and B are the complex field amplitudes of the forward-propagating and backward-propagating waves, respectively.

The ramping of input parameters such as the wavelength to compute the reflectivity spectrum is discussed in [Wavelength Ramping for Computing Optical Generation on page 484](#). It is also possible to calculate the integrated optical generation that results from a given illumination spectrum. This feature is described in [Spectral Illumination on page 486](#).

NOTE The new transfer matrix solver is only supported with grid files in the TDR format.

For backward compatibility, documentation of the old transfer matrix solver is given here. Refer to the note at the end of this section to find out which of the two solvers best suits your needs.

In the `OpticalGeneration` section inside the `Physics` section, all relevant parameters for the transfer matrix method in a Sentaurus Device simulation are given. The syntax of the old transfer matrix solver is:

```
OpticalGeneration {  
    ...  
    QuantumYield = 1  
    Light (  
        Direction = (0, 1)  
        Polarization = 0.5  
        Wavelength = [0.3] # [um]  
        Intensity = [0.1] # [W/cm^2]  
    )  
    Layers (  
        InitialLayerRefract = 1  
        (  
            Thickness = 1  
            Left = (0, 1) # [um]  
            Right = (5, 1) # [um]  
        )  
        (  
            Thickness = 15  
            Left = (5, 1) # [um]  
            Right = (50, 1) # [um]  
            SiLayer  
        )  
    )  
    Layers (  
        <Layers Options>  
        (  
            <LayerEntry>  
        (  
            <LayerEntry>  
        )  
    )  
}
```

The keywords of `OpticalGeneration` are listed in [Table 173 on page 1142](#). `QuantumYield` allows you to redefine the quantum yield given by [Eq. 439](#).

The incident light is specified in the `Light` section. Thereby, `Direction` denotes the direction of the impinging light. `Polarization` is the ratio of the transversal electric and magnetic incident intensities (see [Eq. 437, p. 440](#)). Its value varies between 0 and 1. `Intensity` and `Wavelength` are used to specify multiple intensity and wavelength values. Both lists must have the same length.

The layer information is specified in the `Layers` section. Multiple `Layers` sections are allowed to take into account lateral variation of the device structure. `InitialLayerRefract` specifies the refractive index of the layer in which the light wave starts. `FacetAngle` is used for the simulation of inverted pyramid structures of solar cells.

Within a `Layers` section, an arbitrary number of layer entries (`LayerEntry`) can be defined. The keywords of `LayerEntry` are summarized in [Table 170 on page 1140](#). Layer entries must be specified in the order of their penetration. The window of illumination is defined by `Left` and `Right` in 2D. The illumination window in 3D is a parallelogram with one corner at `Middle` and the sides given by `(Left-Middle)` and `(Right-Middle)`. `Thickness` defines the thickness of the layer. The keyword `SiLayer` must be used in one layer only. The generation rates are calculated for this layer.

The optical properties are extracted from the `TableODB` section of the parameter file (see [Table-based Optical Properties of Materials in Parameter File on page 475](#)). `Refract` and `Absorption` allow you to specify the refractive index and absorption coefficient that overwrite the values from `TableODB`.

`OpticalGeneration` in the plot `Plot` section allow you to visualize optical generation by the transfer matrix method.

NOTE The new transfer matrix solver offers several advantages compared to the former one and is recommended for most applications. It extracts the layer structure automatically and, therefore, guarantees a consistent simulation setup.

Furthermore, the generation rate can be calculated for structures consisting of several absorptive regions with different band gaps. If desired, the calculation of the optical generation can be suppressed for specific regions or materials. With the new solver, it is also possible to compute an integrated generation rate from an illumination spectrum given in a file. This feature can be used in combination with ramping several input parameters such as wavelength, intensity, polarization, and angle of incidence.

On the other hand, the former transfer matrix solver can be applied to 3D simulations and allows different ways to specify the complex refractive index. Additionally, it offers a model that describes the thermalization process of electron–hole pairs generated under the influence of ultraviolet radiation.

Finite-Difference Time-Domain Method

Sentaurus Device Electromagnetic Wave Solver (EMW) is an electromagnetic solver based on the finite-difference time-domain (FDTD) method. Sentaurus Device can run EMW using an interface to generate the tensor grid native to EMW from the more general grid of Sentaurus Device, in order to generate an input command file of EMW, and to load the output of an optical generation profile of EMW.

This interface is *not* a coupling of Maxwell equations solved by EMW and semiconductor equations. The Maxwell equations are solved separately, completely outside of Sentaurus Device, and the resulting optical generation is loaded into the continuity equation.

Files of EMW Generation

Optical generation loading from an EMW simulation (EMW generation) is activated with the command `EMWGeneration` in the `Physics` section. However, a significant amount of relevant syntax is involved in the `File` section that should be noted beforehand.

To run an EMW generation, two files are critical: the input command file of EMW and the tensor grid corresponding to the device being simulated. Both can be created by using either this Sentaurus Device–EMW interface or existing files that are specified by you in the `File` section.

Creating the Tensor Grid and EMW Input Command File

If no existing files for the Sentaurus Device–EMW interface are specified in `File` section, the interface creates the requisite files by using the following naming scheme.

In the absence of the command `plot = "xxx"` in the `File` section, the generated tensor grid file is named `EMW_grid.ten`. If the command `plot = "xxx"` is present, the tensor grid is called `EMW_xxx_grid.ten`, which is the name of the plot file inserted between `EMW` and `grid`.

14: Optical Generation

Finite-Difference Time-Domain Method

Similarly, the input command file of EMW is named `EMW_input.cmd` or `EMW_XXX_input.cmd` depending on the presence or absence of the plot file specification:

```
File {
    Grid = "mytest_mdr.grd"
    Doping = "mytest_mdr.dat"
    Plot = "with_air"
}
```

The Sentaurus Device–EMW interface with the above `File` section creates `EMW_with_air_grid.ten` and `EMW_with_air_input.cmd`.

User-Defined Input or Tensor Grid

If you want to use your own input file or tensor grid file of EMW, specification of their name is performed in the `File` section. The keywords that specify the files of the Sentaurus Device–EMW interface are listed in [Table 127 on page 1091](#).

If the Sentaurus Device–EMW interface ‘sees’ `EMWinput = "<filename>"` in the `File` section, the interface does not create either an input file of its own or a grid file.

If it ‘sees’ `EMWgrid = "<filename>"`, it creates a grid file of its own, but generates an input command file with which to run EMW, for example:

```
File {
    Grid = "mytest_mdr.grd"
    Doping = "mytest_mdr.dat"
    Plot = "with_air"
    EMWinput = "MyEMWinput.cmd"
}
```

This code runs EMW with `MyEMWinput.cmd` as the command file. It is assumed that, by specifying the input command file, you have also provided the tensor grid file. Another example is:

```
File {
    Grid = "mytest_mdr.grd"
    Doping = "mytest_mdr.dat"
    Plot = "with_air"
    EMWgrid = "MyEMWGrid.ten"
}
```

This code creates an input file of EMW with the information that is provided in the main `EMWGeneration` area of the `Physics` section, but does not generate a tensor grid file.

Log of EMW Run

The log of an execution of EMW is stored in <EMWinputfile.cmd>.log. So, if EMW undergoes an abnormal execution, refer to this file for more information.

Syntax of EMW Generation: Input File of EMW

Optical generation from FDTD simulation is activated by the `EMWGeneration` statement in the `Physics` section. [Table 168 on page 1138](#) lists the keywords of `EMWGeneration`.

The following code excerpt describes the general syntax of EMW generation, whereby `NodePerWavelength` specifies the minimum number of nodes that is placed in a wavelength in all directions and `SmoothingFactor` regulates the grid spacing in the generated tensor mesh (see [SmoothingFactor on page 459](#)):

```
Physics { ...
    EMWGeneration(
        NodePerWavelength = ...
        SmoothingFactor = ...
        Boundary(
            ...
        )
        Excitation(
            ...
        )
        Material(
            ...
        )
        AutoMatGen(
            ...
        )
        GridBoundX = ...
        GridBoundY = ...
        GridBoundZ = ...
    )
}
```

Further details about the use of `EMWGeneration` can be found for:

- Specification of global parameters (see [Globals in Table 168 on page 1138](#) and [Multithreading for EMW Generation: Syntax on page 460](#))
- Specification of different boundary conditions (see [Boundary on page 450](#))
- Definition of electromagnetic sources (see [Excitation on page 451](#))
- Definition of material properties in the command file or automatic extraction of material properties (see [Material/AutoMatGen on page 453](#))

14: Optical Generation

Finite-Difference Time-Domain Method

- Specification of the EMW simulation domain (see [GridBoundX](#), [GridBoundY](#), and [GridBoundZ](#) on page 457)
- Specification of detector parameters for terminating the FDTD time-stepping procedure (see [Detector](#) in [Table 168 on page 1138](#))
- Specification of plot parameters (see [Plot](#) in [Table 168 on page 1138](#))

`EmlabGeneration` in the plot `Plot` section can be used to visualize optical generation from FDTD simulation.

Boundary

Boundary denotes the boundary condition. EMW has options for the following boundary conditions:

- First-order and second-order Mur
- Higdon operator (up to fourth-order)
- Perfectly matched layer (PML)
- Periodic boundary condition

Each of these boundary conditions can be restricted to any subset of the six boundary planes. This allows combinations such as periodicity on one direction and absorbing boundary conditions in the remaining directions. However, in EMW, there is one exception to this general concept of combining different types of boundary condition: PML boundary conditions cannot be combined with other types of boundary condition. The default boundary condition chosen by EMW is the Higdon condition of second-order.

Other choices can be made with the `Boundary` statement in the command file of EMW. The boundary arguments are listed in [Table 168 on page 1138](#).

NOTE As nonnumeric values assigned to the arguments inside `Boundary` or another `EMWGeneration` option are strings that are written exactly in the input file of EMW that is generated during Sentaurus Device execution, there is no Sentaurus Device check for their validity while parsing Sentaurus Device input. EMW registers an error upon its execution if illegal input is used. In addition, as these values are strings, they must be in quotation marks ("").

An example is:

```
Physics { ...
    EMWGeneration(
        Boundary(
            Type = "Periodic"
            # notice the quotation marks("") for Type argument is assigned
```

```

        # a string value.
        Side = "X"
    )
Boundary(
    Type = "Higdon"
    Side = "Y"
    Order = 2
)
Excitation(
    <options>
)
)
}
}
```

Excitation

Excitation describes the electromagnetic source. Any number of excitations is allowed. At least one Excitation section is necessary for Sentaurus Device to generate a tensor grid. WaveLength of the first excitation is used to compute wavelength-dependent material parameters. The existing interface syntax describes a plane wave striking the device being simulated. WaveLength of the plane wave must be specified and it also determines the minimum grid space when the tensor grid is being generated. The unit of wavelength for the Sentaurus Device-EMW interface is meter.

For 2D geometry, the direction of the plane wave propagation is defined by Theta, the angle the propagation direction makes with the positive y-axis. The range of Theta is from 0 to 180. Phi and Psi should not be specified for 2D.

For 3D geometry, the direction of the plane wave propagation uses three angles: Theta, Phi, and Psi. Theta has different meanings in 2D and 3D than in 2D, where it is the angle the wave propagation direction makes with the positive z-axis. The range of Theta is still from 0 to 180. Phi is the angle that the wave direction makes with the positive x-axis. Phi can be a number between 0 and 360. Psi is the polarization angle in degrees of the plane wave measured from the direction given by $k \times z$, where k is the wave vector and z is a vector in the z-direction.

Power in the plane wave can be specified as WavePower [W/m^2] or Amplitude of the electric field [V/m]. When both are specified, Amplitude is ignored.

NOTE The option WavePower is not available in Sentaurus Device Electromagnetic Wave Solver (EMW).

14: Optical Generation

Finite-Difference Time-Domain Method

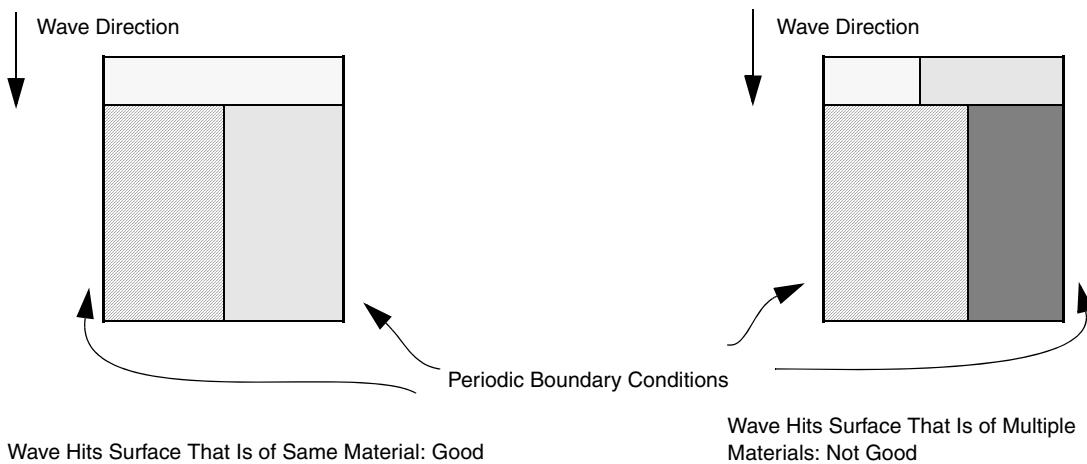
Electric field amplitude A is related to wave power P of the plane wave by:

$$A = \sqrt{2Z_0P} \quad (448)$$

where $Z_0 = \sqrt{\frac{\mu_0}{\epsilon_0}} = 376.73031 \Omega$.

NOTE The excitation default in the Sentaurus Device–EMW interface relies on automatic, which is the excitation region option in EMW. Of most importance to the Sentaurus Device–EMW user is that the ‘surfaces’ that the plane wave hits must be of uniform material (the material can also be a vacuum).

[Figure 47](#) shows a device with periodic boundary conditions in the y-direction. There are two surfaces of interest in this scheme: top x surface and bottom x surface. If the wave is descending, it hits the top x surface, which consists of the same material. For devices without periodic boundary conditions, more than one surface may need to be considered.



[Figure 47](#) Device with periodic boundary conditions in y-direction

Syntax is available that will surround an existing device with a vacuum and, therefore, create a uniform material surface (see [EMW Generation: Tensor Grid, Syntax, and Algorithm on page 456](#)).

[Table 168 on page 1138](#) lists the keywords that are used to define in the `Excitation`.

Excitation Example

```

Physics {
    EMWGeneration(
        Boundary(
            Type = "Periodic"
            # notice the quotation marks("") for Type argument is assigned
            # a string value.
            Side = "X"
        )
        Boundary(
            Type = "Higdon"
            Side = "Y"
            Order = 2
        )
        Excitation(
            WavePower = 1e4 # [W/m^2]
            WaveLength = 1e-7# [m]
            Theta = 180
        )
    )
}

```

This 2D example with its combination of boundary conditions and excitation will create a plane wave ‘descending’ on a device with a power of 1×10^4 W/m².

Plane wave excitation is not the only excitation option available in EMW. Full EMW features are accessed by you by creating your own input file of EMW and providing the file name to Sentaurus Device.

Material/AutoMatGen

EMW has its own database of material properties, MATDB. The most common materials used in the semiconductor field are listed. However, if a material is required that is not in MATDB or you do not want to use the values of existing material properties, there is the option to use choice values for the material properties.

Material properties can be specified by either automatically extracting relative permittivity and conductivity from the optical properties of the absorption coefficient and refractive index, or manually entering the material properties in the input file.

Where the main interest is in the optical generation, the material properties relevant to an interface are permittivity, permeability, conductivity, and name. The name given in the Material section should match the name in the grid file of Sentaurus Device.

[Table 168 on page 1138](#) lists the arguments for the Material section.

14: Optical Generation

Finite-Difference Time-Domain Method

An example that overwrites the material properties of silicon is shown below (parameters of exotic materials that are not contained in MATDB can be defined in a similar way):

```
Physics { ...
    EMWGeneration(
        Boundary(
            Type = "Periodic"
            # notice the quotation marks("") for Type argument is assigned
            # a string value.
            Side = "X"
        )
        Boundary(
            Type = "Higdon"
            Side = "Y"
            Order = 2
        )
        Excitation(
            WavePower = 1e4 # [W/m^2]
            WaveLength = 1e-7# [m]
            Theta = 180
        )
        Material(
            Name = "Silicon"
            Permittivity = 11.7
            Permeability = 1
            Conductivity = 9671.78
        )
    )
}
```

There is a more convenient way of specifying unknown material properties or overwriting existing ones that are unsatisfactory. The AutoMatGen section extracts permittivity and conductivity from the internal parameters of Sentaurus Device. The keywords of AutoMatGen are summarized in [Table 161 on page 1132](#).

RefractiveIndex and SemAbs specify a refractive index and absorption coefficient model, respectively. Then, permittivity and conductivity are extracted from the selected models. The permeability is assumed to be 1. All models that are described in [Refractive Index Models on page 472](#) and [Absorption Models on page 473](#) are valid for optical generation from FDTD.

Permittivity ϵ' is related to the refractive index n and the extinction coefficient k according to:

$$\epsilon' = n^2 - k^2 \quad (449)$$

Conductivity σ is related to absorption coefficient α , and refractive index n by the following relationship:

$$\sigma = \frac{\alpha n}{Z_0} \quad (450)$$

where $Z_0 = \sqrt{\frac{\mu_0}{\epsilon_0}} = 376.73031 \Omega$.

Metal, for which absorption and refractive index models do not exist, takes permittivity and permeability to be 1, and conductivity to be 1×10^{30} .

NOTE A side effect of the matching between the absorption coefficient and refractive index of AutoMatGen and those of RayTracing is that the unit of the constant absorption coefficient is cm^{-1} , not m^{-1} .

The following example demonstrates the use of AutoMatGen:

```

Physics {
    EMWGeneration(
        Boundary(
            Type = "Periodic"
            # notice the quotation marks("") for Type argument is assigned
            # a string value.
            Side = "X"
        )
        Boundary(
            Type = "Higdon"
            Side = "Y"
            Order = 2
        )
        Excitation(
            WavePower = 1e4 # [W/m^2]
            WaveLength = 1e-7# [m]
            Theta = 180
        )
        AutoMatGen(
            SemAbsorption(model = parameter)
            RefractiveIndex(model = ODB)
        )
    )
}

```

14: Optical Generation

Finite-Difference Time-Domain Method

This example uses the parameter absorption coefficient model and refractive index taken from TableODB to write out a Material section in the input command file for EMW that looks like:

```
Material {
    Name = Metal
    Permittivity = 1
    Permeability = 1
    Conductivity = 1e+30
    MagneticConductivity = 0
    Density = 0
    ThermalConductivity = 0
    SpecificHeat = 0
}
Material {
    Name = Si3N4
    Permittivity = 7.5
    Permeability = 1
    Conductivity = 7681.36
    MagneticConductivity = 0
    Density = 0
    ThermalConductivity = 0
    SpecificHeat = 0
}
Material {
    Name = Silicon
    Permittivity = 11.7
    Permeability = 1
    Conductivity = 9671.78
    MagneticConductivity = 0
    Density = 0
    ThermalConductivity = 0
    SpecificHeat = 0
}
```

EMW Generation: Tensor Grid, Syntax, and Algorithm

NOTE You are advised to use Sentaurus Mesh to generate the tensor mesh required for FDTD simulations because the quality of the mesh can be better controlled.

Before describing the syntax relating to tensor grid generation, a brief description of the algorithm is required. The x-, y-, and z-coordinates of vertices of the original Sentaurus Device mesh are queued separately for consideration.

First, the x-coordinates of the vertices are considered. As each x-coordinate of a vertex is popped off the queue of all x-coordinates, some x-coordinates are placed in a second queue. If the location of the x-coordinate is further than `d_min` from other coordinates already in the second queue, it is put in the second queue or it is discarded. Any gaps between the members of this sparser second queue is filled by inserting more x-coordinates, so that no member of this second queue is further than `d_max` from its neighbors.

The maximum grid space, `d_max`, in the resulting tensor mesh is determined by the wavelength specified in the `Excitation` section: `d_max` is `wavelength/20` and `d_min` is `d_max/2`. `GridBound*` arguments can limit the range of coordinates, and the `smoothingfactor` argument can make the distribution of coordinates more uniform. Their functionality and syntax are discussed in the following sections.

The coordinates in this second queue denote the locations of the vertices of the tensor grid in the x-direction. The same algorithm is performed for the y-coordinates and z-coordinates.

GridBoundX, GridBoundY, and GridBoundZ

Arguments relating to the tensor grid that Sentaurus Device generates for EMW are `GridBound*` series arguments that specify the domain that is represented by the tensor grid that is generated and, therefore, the physical domain where the EMW simulation will run. `GridBound*` is followed by a two-entry vector in parentheses, for example, `GridBoundX = (1, 5)` binds the x-grid in the interval [1, 5].

[Figure 48 on page 458](#) demonstrates this concept. Similarly, `GridBoundZ` ‘bind’ the resulting EMW device in the z-direction. If `GridBoundY` is not specified, as in the second figure, the entire Sentaurus Device device in the y-direction will be present in the EMW device and nothing else.

The region that does not correspond to any part of the original device is considered to be a vacuum. The `GridBound*` series of arguments has two functions. First, by binding the simulation area to a smaller region where all the interesting events will occur, the Sentaurus Device–EMW interface can speed up the EMW run. Second, for the excitation option used by the Sentaurus Device–EMW interface, the very ‘top’ surface where the plane wave will first come in contact with the device needs to be uniform in material. By ‘padding’ the top with vacuum for some uneven-surfaced devices, such as a silicon dioxide lens, there can be a layer of uniform material covering the top surface without you having to alter the original device. Further, it may be desirable to have a vacuum layer where the excitation plane wave of devices, such as solar cells and photo detectors, will probably come from.

14: Optical Generation

Finite-Difference Time-Domain Method

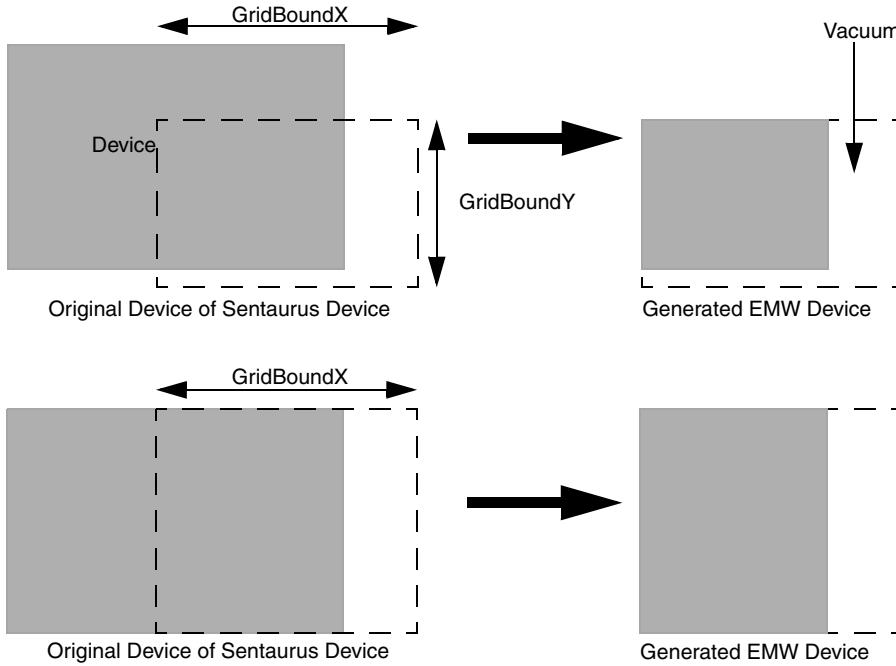


Figure 48 Illustration of GridBoundX, GridBoundY, and GridBoundZ

An example of code is:

```
Physics { ...
    EMWGeneration(
        Boundary(
            Type = "Periodic"
            # notice the quotation marks("") for Type argument is assigned
            # a string value.
            Side = "X"
        )
        Boundary(
            Type = "Higdon"
            Side = "Y"
            Order = 2
        )
        Excitation(
            WavePower = 1e4 # [W/m^2]
            WaveLength = 1e-7# [m]
            Theta = 180
        )
        AutoMatGen(
            SemAbsorption(model = parameter)
            RefractiveIndex(model = ODB)
        )
    )
}
```

```

    SmoothingFactor = 1.5
    GridBoundX = (0, 10)
    GridBoundY = (-1, 12)
)
}

```

SmoothingFactor

The argument `smoothingfactor`, which relates to the tensor grid that Sentaurus Device generates for EMW, regulates the grid spacing in the generated tensor mesh. Closer to uniform, a mesh with grid spacing that is equal, a tensor mesh is, more stable the FDTD simulation is. Although complete uniformity is not necessary, a large difference between grid spacing in adjacent nodes is not good. The argument `smoothingfactor = s` in the `EMWGeneration` statement forces the ratio of adjacent grid spacings in the generated mesh within the smoothing factor `s`, which is a number strictly greater than 1.1. If the smoothing factor is less than 1.1, Sentaurus Device issues a warning and uses 1.1.

The following is a sample code containing the `Smoothing` option of `EMWGeneration`:

```

Physics { ...
    EMWGeneration(
        Boundary(
            Type = "Periodic"
            # notice the quotation marks("") for Type argument is assigned
            # a string value.
            Side = "X"
        )
        Boundary(
            Type = "Higdon"
            Side = "Y"
            Order = 2
        )
        Excitation(
            WavePower = 1e4 # [W/m^2]
            WaveLength = 1e-7# [m]
            Theta = 180
        )
        AutoMatGen(
            SemAbsorption(model = parameter)
            RefractiveIndex(model = ODB)
        )
        Smoothingfactor = 1.5
    )
}

```

Multithreading for EMW Generation: Syntax

To speed up EMW simulations on shared-memory architectures, support for multithreading can be switched on. In general, the number of threads to be used in an EMW simulation can be specified either in the `Globals` section of the command file or by an environment variable. If both are specified, the value in the command file overwrites the value of the environment variable. The following two examples illustrate the use of both options to set the number of threads to two.

The specification in the `Globals` section is:

```
Globals {  
    NumberOfThreads = 2  
}
```

The specification using a UNIX environment variable (for example, C shell syntax) is:

```
setenv EMW_NUMBER_OF_THREADS 2
```

By default, the number of threads is set to one. A fall-back to non-multithreading simulations is also available and can be activated in the `Globals` section of the command file:

```
Globals {  
    ForceSerialExecution = yes  
}
```

This avoids any use of multithreading methods and is equivalent to previous versions of EMW that do not support multithreading.

EMW Interface to Hardware Acceleration

An interface from EMW has been built to access the hardware acceleration of FDTD by Acceleware. It is called EMW-X, an abbreviation of EMW-Acceleware. The Acceleware hardware performs the basic FDTD computation step. EMW still controls the time-stepping and steady-state detection of the simulation, besides tensor mesh translation, variables mapping, results extraction, and computation of optical generation and absorbed power (see [Sentaurus Device Electromagnetic Wave Solver User Guide, Chapter 6](#) on page 69).

The options to initialize the boundary conditions, set the excitation source, control steady-state detection error, specify the file name of the tensor grid file, and so on must be input in a separate command file unique to EMW-X. The details of EMW-X and its unique command file are described in [Sentaurus Device Electromagnetic Wave Solver User Guide, EMW-X Command File](#) on page 72.

The tensor grid is created by Sentaurus Mesh, and it must include excitation boxes to define the excitation plane wave (see [Sentaurus Device Electromagnetic Wave Solver User Guide, Generating Tensor Mesh on page 72](#)).

After the EMW-X command file and the tensor grid file have been created, you can activate the hardware acceleration through Sentaurus Device using the EMW–Sentaurus Device framework. The syntax is:

```
File {...  
    EMWIInput = "emwx_command.cmd"  
}  
  
Physics {...  
    EMWGeneration(  
        AccelewareOption  
    )  
}  
  
Solve {...  
    Quasistationary(...){...}  
}
```

Some comments about the syntax:

- You must create the `emwx_command.cmd` file separately using keywords that are specific only to EMW-X (see [Sentaurus Device Electromagnetic Wave Solver User Guide, EMW-X Command File on page 72](#)).
- Only the keyword `AccelewareOption` is required in the `EMWGeneration` section. When this keyword is detected, the other options are ignored. The purpose of this is to keep the options of EMW-X separate from Sentaurus Device to achieve modularity.
- A system call is performed to call EMW-X to compute the optical generation. After the computation is completed, the command is returned to Sentaurus Device together with the optical generation profile for the device, and subsequent commands in the `Solve` section are executed.

NOTE After the optical generation is computed by EMW-X, it is saved to a file. It is important to specify the file name as `EMW_Extract_generation_emw.dat` in the EMW-X command file because this is the file that Sentaurus Device will try to load after calling EMW-X.

14: Optical Generation

Beam Propagation Method

To specify the correct optical generation file name, include the following statements in `emwx_command.cmd`:

```
Extractor {
    Quantity      = {OpticalGeneration, PowerFluxDensity}
    Name          = "EMW_Extract_generation"
    Type          = dfgeo
    DfgeoFile     = "devicegrid.grd"    # this should be the SDevice grid
    OutputFormat  = binary
}
```

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 The use of BPM for 3D CMOS image sensors is discouraged because the 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 [5].

The solution of the scalar Helmholtz equation:

$$\left[\nabla_t^2 + \frac{\partial^2}{\partial z^2} + k_0^2 n^2(x, y, z) \right] \phi(x, y, z) = 0 \quad (451)$$

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:

$$\frac{\partial}{\partial z} \phi(x, y, z + \Delta z) = \mp i \sqrt{k_0^2 n^2 + \nabla_t^2} \phi(x, y, z) = \pm i \wp \phi(x, y, z) \quad (452)$$

In the paraxial approximation, the operator \wp reduces to:

$$\wp \approx \sqrt{k_0^2 n_0^2 + \nabla_t^2} + k_0 \delta n \quad (453)$$

where n_0 is taken as a constant reference refractive index in every transverse plane. By expressing the field ϕ at z as a spatial Fourier decomposition of plane waves, the solution to [Eq. 451](#) for forward-propagating waves reads:

$$\phi(x, y, z + \Delta z) = \frac{1}{(2\pi)^2} \exp(i k_0 \delta n \Delta z) \int_{-\infty}^{\infty} d\vec{k}_t \exp(i \vec{k}_t \cdot \vec{r}) \exp(i \sqrt{k_0^2 n_0^2 - \vec{k}_t^2} \Delta z) \tilde{\phi}(\vec{k}_t, z) \quad (454)$$

where $\tilde{\phi}$ denotes the transverse spatial Fourier transform. As can be seen from [Eq. 454](#), 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 [Eq. 454](#), an FFT algorithm is used to compute the forward and inverse Fourier transform. The refractive index profile is determined by the values of the TableODB section in the parameter file (see [Table-based Optical Properties of Materials in Parameter File on page 475](#)).

Bidirectional BPM

The bidirectional algorithm is based on two operators as described by KaczmarSKI and Lagasse [\[6\]](#). 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.

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

14: Optical Generation

Beam Propagation Method

Device supports the PML boundary condition, which has been developed for continuum spectra.

The formulation is based on the introduction of a stretching operator:

$$\nabla_s \equiv \frac{1}{S_x} \hat{x} \frac{\partial}{\partial x} + \frac{1}{S_y} \hat{y} \frac{\partial}{\partial y} + \frac{1}{S_z} \hat{z} \frac{\partial}{\partial z} \quad (455)$$

from which an equivalent set of the Maxwell equations can be derived. In Eq. 455, 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:

$$\phi(x, y, z + \Delta z) = \frac{1}{(2\pi)^2} \exp(iS_z k_0 \delta n \Delta z) \int \int_{-\infty}^{\infty} d\vec{k}_t \exp(i\vec{k}_t \cdot \vec{r}) \exp(iS_z) \exp(i\sqrt{k_0^2 n_0^2 - \vec{k}_t^2} \Delta z) \tilde{\phi}(\vec{k}_t, z) \quad (456)$$

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

The command file syntax for optical generation by BPM differs from the syntax of previous methods in that the specification of the optical solver parameters is given in a separate `Optics` section within the `Physics` section. Consequently, the `OpticalGeneration` section contains only parameters directly related to the calculation of the optical generation from the optical intensity.

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` parameter, as defined in Eq. 438 and Eq. 439, can be specified. In the `Optics` section, the scalar BPM solver and its parameters are specified. 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 Eq. 453. Parameters related to the excitation field and the boundary conditions for each spatial dimension are grouped in subsections as explained below:

```
Physics {
    ...
    OpticalGeneration (
        QuantumYield = 1.0
    )
    Optics(

```

```
BPMScalar (
    GridNodes = (256,256,1600)
    ReferenceRefractiveIndex = 2.2
    Excitation (
        ...
    )
    Boundary (
        ...
    )
)
...
}
```

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 may 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 164 on page 1134](#).

Bidirectional BPM

The bidirectional BPM solver is activated by specifying a `Bidirectional` section in the `BPM` section:

```
Optics (
    BPMScalar (
        ...
        Bidirectional (
            Iterations = 3
            Error = 1e-3
        )
    )
)
```

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.

NOTE The bidirectional beam propagation method is only supported with grid files in TDR format.

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 49](#).

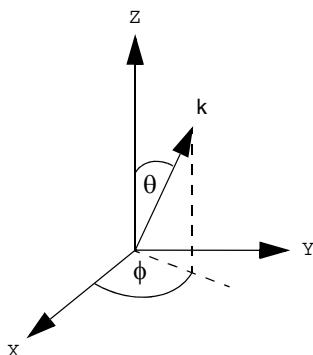


Figure 49 Definition of angles for specification of propagation direction in three dimensions

The incident light power in the plane wave can be specified as the `WavePower` [W/m^2] or `Amplitude` of the electric field [V/m] (the conversion between these options is given in [Eq. 448](#)). For a Gaussian excitation, the given value refers to its maximum.

The excitation parameters common to all types of excitation are:

```
Physics {  
  ...  
  Optics(  
    BPMScalar (  
      Excitation (  
        Theta = 10.0 # [degree]  
        Phi = 60.0 # [degree], only in 3D  
        Wavelength = 0.3 # [\mu m]  
        Amplitude = 1.0 # [V/m]  
      ...  
    )  
  )  
}
```

```
)  
...  
)  
}  
}
```

A Gaussian excitation along the propagation direction defined by k is determined by its half-width `SigmaGauss` [μm] and its center position `CenterGauss` [μm] as shown below for a 2D and 3D device geometry.

Gaussian excitation (2D):

```
Physics {  
...  
Optics(  
BPMScalar (  
Excitation (  
...  
Type = "Gaussian"  
SigmaGauss = (2.0) # [ $\mu\text{m}$ ]  
CenterGauss = (0.0) # [ $\mu\text{m}$ ]  
)  
...  
)  
}  
}
```

Gaussian excitation (3D):

```
Physics {  
...  
Optics(  
BPMScalar (  
Excitation (  
...  
Type = Gaussian  
SigmaGauss = (2.0,4.0) # [ $\mu\text{m}$ ]  
CenterGauss = (0.0,1.0) # [ $\mu\text{m}$ ]  
)  
}  
}  
}
```

14: 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:

$$F_{xy} = \left[1 + \exp\left(\frac{|\tilde{x}| - x_0}{s_x}\right) \right]^{-1} \cdot \left[1 + \exp\left(\frac{|\tilde{y}| - y_0}{s_y}\right) \right]^{-1} \quad (457)$$
$$\phi(x, y, z=0) = F_{xy} \cdot \phi_0$$

which can be tuned by adjusting the TruncationSlope and TruncationPosition parameters in the Excitation section. In Eq. 457, ϕ_0 is the amplitude of the incident light, $s_{x,y}$ denotes the TruncationSlope, \tilde{x}, \tilde{y} is the position with respect to the symmetry point, and x_0, y_0 is the half width. Both \tilde{x}, \tilde{y} and x_0, y_0 are deduced from the TruncationPosition parameters. In two dimensions, the second factor on the right-hand side in Eq. 457 is omitted.

Truncated plane wave excitation (2D):

```
Physics {
    ...
    Optics(
        BPMScalar (
            Excitation (
                ...
                    Type = "PlaneWave"
                    TruncationPositionX = (-1.0,1.0)
                    TruncationSlope = (0.3)
                )
                ...
            )
        )
    }
}
```

Truncated plane wave excitation (3D):

```
Physics {
    ...
    Optics(
        BPMScalar (
            Excitation (
                ...
                    Type = "PlaneWave"
                    TruncationPositionX = (-1.0,1.0)
                    TruncationPositionY = (-2.0,3.0)
                    TruncationSlope = (0.3,0.3)
                )
                ...
            )
        )
    }
}
```

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(
        BPMScalar (
            ...
            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)
            )
            ...
        )
    )
}
```

Optics Stand-Alone Option

Similar to the optical solvers used in laser simulations described in [Optics Stand-alone Option on page 795](#), it is possible to run the BPM solver without having to perform an electrical device simulation. This feature allows for an efficient and separate study of the optical problem in the initial design phase.

To activate the optics stand-alone option, the `Optics` keyword must be specified exclusively in the `Solve` section as shown here. The parameters of the BPM solver are specified in the standard way:

```
Physics {
    Optics(
        BPMScalar (
            Excitation (
                ...
            )
            Boundary (
                ...
            )
            ...
        )
    )
}

Solve {
    Optics
}
```

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 71](#). In the following example, the `Wavelength` of the excitation is ramped from 0.3 μm to 1 μm in 70 nm steps.

Other parameters that can be ramped are the `WavePower` and its equivalents such as the `Amplitude` of the electric field or the `WaveIntensity`:

```
Physics {
    Optics(
        BPMScalar (
            Excitation (
                ...
                Wavelength = 0.3 #[μm]
```

```

        ...
    )
Boundary (
    ...
)
...
)
}
}

Solve {
    Quasistationary (
        InitialStep=0.1 MaxStep=0.1 MinStep=0.1
        Goal {Model="Optics" Parameter="Wavelength" value = 1.0}
    )
    Optics
}
}

```

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"
)

```

14: Optical Generation

Refractive Index Models

```
)  
...  
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  
    Zmax = 8  
) {  
    ...  
}
```

Refractive Index Models

Sentaurus Device offers several possibilities to define the refractive index that is used for the optical generation computation:

- A constant value can be defined for some of the models in the input file of Sentaurus Device. The corresponding keyword can be found in the model description.
- A dependence on the mole fraction and temperature can be taken into account if it is defined in the parameter file of Sentaurus Device.
- A table-based optical property that depends on photon energy can be entered in the parameter file of Sentaurus Device (see [Table-based Optical Properties of Materials in Parameter File on page 475](#)).
- A new expression is created for the refractive index using the PMI.

Depending on the optical generation model and optical solver, only selective options from the above list may be valid (refer to the corresponding model descriptions or optical solvers for more details).

Absorption Models

Sentaurus Device offers several possibilities to define the absorption coefficient that is used for the optical generation computation:

- A constant value can be defined for some of the models in the input file of Sentaurus Device. The corresponding keyword can be found in the model description.
- A dependence on the photon energy, mole fraction, and so on is in the parameter file of Sentaurus Device (see [Default Absorption Model from Parameter File on page 473](#)).
- A table-based optical property that depends on photon energy can be entered in the parameter file of Sentaurus Device (see [Table-based Optical Properties of Materials in Parameter File on page 475](#)).
- An absorption coefficient model for indirect bandgap material (see [Absorption Coefficient Model on page 475](#)).
- A new expression is created for the absorption coefficient using the PMI.

Depending on the optical generation model and optical solver, only selective options from the above list may be valid (refer to the corresponding model descriptions or optical solvers for more details).

NOTE The temperature dependence of the parameter file–based absorption coefficient models, both the default model and absorption coefficient model [7], is dependent on the global device temperature due to convergence reasons. Local temperature–dependent models can be created by using the PMI.

Default Absorption Model from Parameter File

NOTE The absorption models from the parameter file cannot be used in the raytracer, TMM, and unified interface for optical generation. The complex refractive index model should be used instead.

The absorption coefficient has two available models that can be selected in the parameter file of Sentaurus Device. The simpler one is:

$$\alpha(E_{\text{ph}}) = \begin{cases} \alpha_1 \exp(E_{\text{ph}} - E_1/E_2) & \text{for } E_{\text{ph}} < E_1 \\ \alpha_1 + \alpha_2 E_{\text{ph}} - E_1/E_2^P & \text{for } E_{\text{ph}} \geq E_1 \end{cases} \quad (458)$$

14: Optical Generation

Absorption Models

where E_{ph} is the photon energy, and α_1 and α_2 (given in cm^{-1}), E_1 and E_2 (given in eV) and p are the model parameters that can be specified in the parameter file:

```
Absorption
{ Formula = 1
  A1 = <value>
  A2 = <value>
  E1 = <value>
  E2 = <value>
  p = <value>
}
```

Another model [8] for mole-dependent $\text{Hg}_{1-x}\text{Cd}_x\text{Te}$ is generalized as:

$$E_T = E_0 + \frac{T + T_0}{\sigma} \ln \frac{\alpha_T}{\alpha_0}$$

$$\alpha(E_{ph}, T) = \begin{cases} \alpha_0 \exp\left(\frac{\sigma(E_{ph} - E_0)}{T + T_0}\right) & \text{for } E_{ph} < E_T \\ \alpha_T \left(\frac{2\sigma}{T + T_0} \left(E_{ph} - E_0 - \frac{T + T_0}{\sigma} \left(\ln \frac{\alpha_T}{\alpha_0} - 0.5 \right) \right) \right)^{0.5} & \text{for } E_{ph} \geq E_T \end{cases} \quad (459)$$

where α_T and α_0 (given in cm^{-1}), E_0 (given in eV), T_0 (given in K), and σ (given in K/eV) are model parameters that can be specified in the parameter file:

```
Absorption
{ Formula = 2
  AT = <value>
  A0 = <value>
  E0 = <value>
  T0 = <value>
  S = <value>
}
```

All parameters of both models can have a mole dependence for mole-dependent materials using the standard Sentaurus Device technique with linear interpolation on specified mole intervals. However, for the material $\text{Hg}_{1-x}\text{Cd}_x\text{Te}$, [8] provides the following mole dependencies:

$$\begin{aligned} \alpha_0/\text{cm}^{-1} &= \exp(-18.88 + 53.61x) \\ \alpha_T/\text{cm}^{-1} &= 100 + 5000x \\ \sigma/(\text{K/eV}) &= 3.267 \times 10^4 (1 + x) \\ E_0/\text{eV} &= -0.3424 + 1.838x \\ T_0/\text{K} &= 81 \end{aligned} \quad (460)$$

where x denotes the CdTe mole fraction. The parameter α_0 has an exponential dependence that cannot be described by the linear interpolation used for mole-dependent parameters in Sentaurus Device. To activate the above expressions, the following must be present in the parameter file:

```
Absorption { Formula = 2 HgCdTe }
```

Table-based Optical Properties of Materials in Parameter File

Table entry of real refractive index, n , and extinction coefficient, k , versus wavelength. Absorption coefficient is computed from k by $\alpha(\lambda) = \frac{2\pi k}{\lambda}$.

Sentaurus Device has the tabled values of these optical properties for silicon, silicon dioxide, aluminum, gold, silver, platinum, silicon nitride, tungsten, and air (gas). For other materials, the TableODB section can be inserted in the parameter file of Sentaurus Device, and you can create a table for a material or a region.

The first element in a row is the wavelength, followed by n and k . The row is terminated by a semicolon. There can be any number of rows greater than 3, but not less, for a cubic spline to be formed from the table entries.

```
TableODB
{
  *WAVELENGTH    n        k
    0.051    0.804    0.322;
    0.053    0.811    0.366;
    0.055    0.822    0.408;
    0.056    0.829    0.43;
    0.058    0.843    0.47;
}
```

Absorption Coefficient Model

An absorption model [7] is implemented in Sentaurus Device for silicon and materials with similar bandgap structure. The model provides an absorption coefficient model and fits absorption data of silicon supplied by NASA, by varying temperature and photon energy. The model is switched on by inserting the statement `SemAbsorption` in the appropriate `Physics` section.

NOTE The placement of this statement above is outside either `RayTrace` or `OptBeam` where it has traditionally resided.

14: Optical Generation

Absorption Models

The resulting equation with absorption coefficient α is:

$$\alpha(T) = \sum_{i=1}^2 \sum_{j=1}^2 C_i A_j \left[\frac{\{hf - E_{gj}(T) + E_{pi}\}^2}{e^{\frac{-E_{pi}}{kT}} - 1} + \frac{\{hf - E_{gj}(T) - E_{pi}\}^2}{1 - e^{\frac{-E_{pi}}{kT}}} \right] + A_d [hf - E_{gd}(T)]^{\frac{1}{2}} \quad (461)$$

where $E_g(T) = E_g(0) - [\beta T^2 / (T + \gamma)]$ and T is the temperature [K]. [Table 74](#) lists the default parameter values for silicon used in Sentaurus Device as published [7].

Table 74 Default parameter values of silicon

Quantity	Value	Comment
Eg1(0)	1.1557 [eV]	Indirect gap
Eg2(0)	2.5 [eV]	Indirect gap
Egd(0)	3.2 [eV]	Direct allowed gap
Ep1	1.827×10^{-2} [eV]	TA, Theta = 212 K
Ep2	5.773×10^{-2} [eV]	TO, Theta = 670 K
C1	5.5	–
C2	4.0	–
A1	3.231×10^2 [$\text{cm}^{-1}/\text{eV}^2$]	–
A2	7.237×10^3 [$\text{cm}^{-1}/\text{eV}^2$]	–
Ad	1.052×10^6 [$\text{cm}^{-1}/\text{eV}^2$]	–
Beta	7.021×10^{-4} [eV/K]	–
Gamma	1108 [K]	–

Further, these parameter values can be changed for some materials with bandgap structures similar to silicon or for silicon itself, by using the following (appropriate for parameter files of Sentaurus Device) and replacing the parameters with desired values:

```
RSSAbsorption
{
  * K. Rajkanan, R. Singh, and J. Shewchun,
  * Absorption Coefficient of Silicon for Solar Cell Calculations
  * Solid-State Electronics 1979 Vol 22. pp793-795
    Egone0 = 1.1557 # [eV]
    Egtwo0 = 2.5    # [eV]
    Egd0 = 3.2     # [eV]
    Ep1 = 0.01827  # [eV]
    Ep2 = 0.05773  # [eV]
    C1 = 5.5       # [1]
    C2 = 4          # [1]
    A1 = 3.2310e+02 # [cm^-1 eV^-2]
```

```

A2 = 7.2370e+03 # [cm^-1 eV-2]
Ad = 1.0520e+06 # [cm^-1 eV-2]
RssBeta = 7.237e-4 # [eV/K]
RssGamma = 1108 # [K]
}

```

If the RSSAbsorption model is toggled in Physics and if the above RSSAbsorption model parameters are not specified in the parameter file, then the parameter values of silicon are used.

The mole fraction dependence of the RSSAbsorption model is identical to that of the default parameter-based absorption model in Sentaurus Device (see [Default Absorption Model from Parameter File on page 473](#)).

The absorption model specification in the Physics section overwrites that in the OptBeam section.

NOTE The absorption coefficient models cannot be used for the raytracer, TMM, and unified interface for optical generation. The complex refractive index model should be used instead.

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. It is available in TDR format only.

This complex refractive index model is designed primarily for use in laser and LED simulations, the unified interface for optical generation, the raytracer, and the TMM optical solvers.

Physical Model

The complex refractive index \tilde{n} can be written as:

$$\tilde{n} = n + i \cdot k \quad (462)$$

with:

$$n = n_0 + \Delta n_\lambda + \Delta n_T + \Delta n_{\text{carr}} + \Delta n_{\text{gain}} \quad (463)$$

$$k = k_0 + \Delta k_\lambda + \Delta k_{\text{carr}} \quad (464)$$

14: Optical Generation

Complex Refractive Index Model

The real part n is composed of the base refractive index n_0 , and the correction terms Δn_λ , Δn_T , Δn_{carr} , and Δn_{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 Δk_λ and Δk_{carr} . The correction terms include the dependency on wavelength and carrier density. The absorption coefficient α is computed from k and wavelength λ according to:

$$\alpha = \frac{4\pi}{\lambda} \cdot k \quad (465)$$

Wavelength Dependency

The complex refractive index model offers three ways to take wavelength dependency into account:

- An analytic formula considers a linear and square dependency on the wavelength λ :

$$\begin{aligned}\Delta n_\lambda &= C_{n,\lambda} \cdot \lambda + D_{n,\lambda} \cdot \lambda^2 \\ \Delta k_\lambda &= C_{k,\lambda} \cdot \lambda + D_{k,\lambda} \cdot \lambda^2\end{aligned} \quad (466)$$

The parameters $C_{n,\lambda}$, $D_{n,\lambda}$, $C_{k,\lambda}$, and $D_{k,\lambda}$ are adjusted in the ComplexRefractiveIndex section of the parameter file (see [Table 77 on page 481](#)).

- Tabulated values can be read from the parameter file. The table contains three columns specifying wavelength λ , 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. At least three rows are required to form a cubic spline from the table entries.
- 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:

$$\Delta n_T = n_0 \cdot C_{n,T} \cdot (T - T_{\text{par}}) \quad (467)$$

The parameters $C_{n,T}$ and T_{par} can be adjusted in the ComplexRefractiveIndex section of the parameter file (see [Table 78 on page 481](#)).

Carrier Dependency

The change in the real part of the complex refractive index due to free carrier absorption is modeled according to [9]:

$$\Delta n_{\text{carr}} = -C_{n, \text{carr}} \cdot \frac{q^2 \lambda^2}{8\pi^2 c^2 \epsilon_0 n_0} \cdot \left(\frac{n}{m_n} + \frac{p}{m_p} \right) \quad (468)$$

where:

- $C_{n, \text{carr}}$ is a fitting parameter.
- q is the elementary charge.
- λ is the wavelength.
- c is the speed of light in free space.
- ϵ_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.

A linear model is available to take into account the change of the extinction coefficient due to free carrier absorption:

$$\Delta k_{\text{carr}} = \frac{\lambda}{4\pi} \cdot (C_{k, \text{carr}, n} \cdot n + C_{k, \text{carr}, p} \cdot p) \quad (469)$$

The parameters $C_{k, \text{carr}, n}$ and $C_{k, \text{carr}, p}$ are fitting parameters.

The parameters $C_{n, \text{carr}}$, $C_{k, \text{carr}, n}$, and $C_{k, \text{carr}, p}$ are adjusted in the ComplexRefractiveIndex section of the parameter file (see [Table 79 on page 481](#)).

Gain Dependency

The complex refractive index model offers two formulas to take into account the gain dependency:

- The linear model is given by:

$$\Delta n_{\text{gain}} = C_{n, \text{gain}} \cdot \left(\frac{n+p}{2} - N_{\text{par}} \right) \quad (470)$$

- The logarithmic model reads:

$$\Delta n_{\text{gain}} = C_{n, \text{gain}} \cdot \log \left(\frac{n+p}{2N_{\text{par}}} \right) \quad (471)$$

14: Optical Generation

Complex Refractive Index Model

The parameters $C_{n, \text{gain}}$ and N_{par} are adjusted in the ComplexRefractiveIndex section of the parameter file (see [Table 80 on page 481](#)).

Using Complex Refractive Index

The complex refractive index model is activated by using the ComplexRefractiveIndex statement in the Physics section. [Table 206 on page 1165](#) lists the available keywords. Therefore, you can select the dependencies that change the complex refractive index, for example:

```
Physics {  
    ComplexRefractiveIndex (  
        WavelengthDep (imag)  
        TemperatureDep (real)  
        CarrierDep (real)  
        GainDep (real(log))  
    )  
}
```

All parameters valid for the ComplexRefractiveIndex section of the parameter file are summarized in [Table 75](#) to [Table 80 on page 481](#). They can be specified for mole-dependent materials using the standard Sentaurus Device technique with linear interpolation on specified mole intervals.

[Table 75](#) lists the base parameters.

Table 75 Base parameters

Symbol	Parameter name	Unit
n_0	n0	1
k_0	k0	1

The keyword `Formula` is used in the parameter file to select one of the three models that are available to describe the wavelength dependency.

[Table 76](#) Model selection for wavelength dependency

Keyword	Value	Description
Formula	0	Use analytic formula (default).
	1	Read tabulated values from parameter file.
	2	Read tabulated values from external file.
	3	Use the tabulated values from the TableODB.

Table 77 lists the parameters that are used to describe wavelength dependency.

Table 77 Parameters for wavelength dependency

Symbol	Parameter name	Unit
$C_{n,\lambda}$	Cn_lambda	μm^{-1}
$D_{n,\lambda}$	Dn_lambda	μm^{-2}
$C_{k,\lambda}$	Ck_lambda	μm^{-1}
$D_{k,\lambda}$	Dk_lambda	μm^{-2}

Table 78 lists the parameters that are used to describe temperature dependency.

Table 78 Parameters for temperature dependency

Symbol	Parameter name	Unit
$C_{n,T}$	Cn_temp	K^{-1}
T_{par}	Tpar	K

Table 79 lists the parameters that are used to describe carrier dependency.

Table 79 Parameters for 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	cm^2	Values for electrons and holes are separated by a comma.

Table 80 lists the parameters that are used to describe gain dependency.

Table 80 Parameters for gain dependence

Symbol	Parameter name	Unit
$C_{n,\text{gain}}$	Cn_gain	cm^3 (linear formula) 1 (logarithmic formula)
N_{par}	Npar	cm^{-3}

14: Optical Generation

Complex Refractive Index Model

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
    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;
        )
    * Example for reading values from external file:
    Formula = 2
    NumericalTable = "GaAs.txt"

    * Example for reading values from TableODB
    Formula = 3

    * Temperature dependence (real):
    Cn_temp = 2.0000e-04      # [K^-1]
    Tpar = 3.0000e+02          # [K]

    * Carrier dependence (real)
    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 complex refractive index is plotted when the keyword ComplexRefractiveIndex is defined in the Plot section.

If TableODB is present but the ComplexRefractiveIndex section is absent in the parameter file, then Formula=3 is used to set the ComplexRefractiveIndex numeric table with the values from TableODB.

The complex refractive index model is available for the following optical solvers:

- Finite-element solver (scalar and vectorial) (see [Finite-Element Formulation on page 762](#)).
- Effective index and transfer matrix method for VCSEL (see [Effective Index Method for VCSELs on page 777](#) and [Transfer Matrix Method for VCSELs on page 775](#)).
- Transfer matrix method for optical generation computation (see [Transfer Matrix Method on page 438](#)).
- Raytracing (see [Raytracing on page 407](#)).

Loading Optical Generation from File

The optical generation profile can be saved as a dataset with the name OpticalGeneration in the output plot files. A previously saved optical generation profile can be loaded using the following syntax in the File section of the command file:

```
File { ...
    OpticalGenerationFile = <filename>
}
```

You can modify the properties of the loaded optical generation profile. Two modifications are possible: scaling the profile and adding a time-transient Gaussian function to the profile. These can be achieved with the following syntax in the Physics section of the command file:

```
Physics { ...
    OpticalGenerationFromFile ( WaveTime = (<t1>,<t2>)
                                WaveTSigma = <float>
                                OptScaling = <float> )
}
```

The entire optical profile can be scaled by the constant number specified by OptScaling. The two parameters ($<t1>$, $<t2>$) in WaveTime determine the interval in time when the profile is constant. Before $<t1>$, the optical generation undergoes a Gaussian increase from zero. After $<t2>$, the optical generation experiences a Gaussian decay to zero.

14: Optical Generation

Wavelength Ramping for Computing Optical Generation

This time function is expressed as:

$$F(t) = \begin{cases} \exp\left(-\sqrt{\frac{t_1-t}{\sigma}}\right) & , t < t_1 \\ 1 & , t_1 \leq t \leq t_2 \\ \exp\left(-\sqrt{\frac{t-t_2}{\sigma}}\right) & , t > t_2 \end{cases} \quad (472)$$

where σ is defined by WaveTSigma.

Wavelength Ramping for Computing Optical Generation

NOTE For a description of parameter ramping using the unified interface for optical generation computation, see [Parameter Ramping on page 406](#).

Sentaurus Device can be used to ramp the values of physical parameters (see [Ramping Physical Parameter Values on page 71](#)). In particular, it is possible to sweep the wavelength of the incident light to extract the spectral dependence of a certain output characteristic conveniently. For a complete list of parameters that can be ramped in the various optical models relevant to computing optical generation, see [Ramping Physical Parameter Values on page 71](#).

Optical solvers such as the beam propagation method (see [Beam Propagation Method on page 462](#)) and the transfer matrix method (see [Transfer Matrix Method on page 438](#)) follow a common syntax:

```
CurrentPlot {  
    Model="Optics" Parameter="Wavelength"  
}  
  
Physics {  
    ...  
    OpticalGeneration (  
        ...  
    )  
    Optics (  
        TMM (  
            ...  
            Excitation (  
                ...  
                Wavelength = 0.3 # [μm]  
            )  
        )  
    )  
}
```

```

Solve {
    Poisson
    Coupled { Poisson Electron Hole }
    Quasistationary (
        Goal { Model="Optics" Parameter="Wavelength" value=0.9 }
    ) {
        Coupled { Poisson Electron Hole }
    }
}

```

In this example, the wavelength is ramped from 0.3 μm to 0.9 μm . The `CurrentPlot` section is needed to plot output quantities as a function of wavelength. For the new transfer matrix solver, the polarization-dependent reflectivity and transmittivity as a function of wavelength for every stripe can be plotted in this way.

The remaining optical solvers are addressed using a specific model in the `Physics` section. The following models also support wavelength ramping:

- `OptBeam` (see [Optical Beam Absorption on page 434](#))
- `RayBeam` (see [Raytracing on page 407](#))
- `OpticalGeneration` (see [Transfer Matrix Method on page 438](#))

To use wavelength ramping for the computation of optical generation:

- Add the wavelength parameter to the `CurrentPlot` section (see [CurrentPlot Section on page 59](#)).
- Declare the desired model for computing optical generation in the `Physics` section. The wavelength that is stated within the model denotes the initial value for the ramp.
- Specify the model name and wavelength parameter in the `Goal` statement of the `Quasistationary` section.

In the following example, `<model_name>` stands for one of the model names `OptBeam`, `RayBeam`, or `OpticalGeneration` (old transfer matrix solver). To ramp the wavelength, `<parameter_name>` must be substituted with `WaveLength` (case sensitive):

```

CurrentPlot {
    Model="<model_name>" Parameter="<parameter_name>"
}

Physics {
    ...
    <model_specification>
}

Solve {
    Poisson

```

14: Optical Generation

Spectral Illumination

```
Coupled { Poisson Electron Hole }
Quasistationary (
    Goal { model=<model_name> parameter=<parameter_name> value=<float> }
)
    Coupled { Poisson Electron Hole }
}
}
```

Spectral Illumination

NOTE For a description of spectral illumination using the unified interface for optical generation computation, see [Illumination Spectrum on page 396](#).

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, the keyword `ComputeFromSpectrum` must be listed in the `Physics` 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 {
    OpticalGeneration (
        ...
        ComputeFromSpectrum
    )
    Optics (
        TMM ( # or RayTrace or BPM
            ...
        )
    )
}

Solve {
    Optics
}
```

or:

```
Solve {
    Quasistationary(...) {...}
}
```

The illumination spectrum file has a two-column format. The first column contains the wavelength in μm and the second column contains the intensity in W/m^2 .

NOTE This feature is only available for the new transfer matrix solver, the beam propagation method, and the raytracer.

The spectral illumination feature can be combined with the wavelength ramping feature. In this case, the integrated generation rate is calculated once at the beginning. For each wavelength addressed in the sweep, the total generation rate is obtained by adding the contribution of the single wavelength source to the integrated generation rate.

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 δP_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 δG^{opt} is an amplitude of a local perturbation. The resulting small-signal device current perturbation δI_{dev} is the sum of real and imaginary parts, and the expressions for the quantum efficiency are:

$$\begin{aligned}\eta &= \frac{Re[\delta I_{\text{dev}}]/q}{\delta P^{\text{tot}}\lambda/hc} \\ C_{\text{opt}} &= \frac{1}{\omega} \cdot \frac{Im[\delta I_{\text{dev}}]/q}{\delta P^{\text{tot}}\lambda/hc} \\ \delta P^{\text{tot}} &= \int_S \delta P_0 ds\end{aligned}\tag{473}$$

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 η and C_{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] A. Liegmann, *The Application of Supernodal Techniques on the Solution of Structurally Symmetric Systems*, Technical Report 92/5, Integrated Systems Laboratory, ETH, Zurich, Switzerland, 1992.
- [2] T.-W. Tang, “Extension of the Scharfetter–Gummel Algorithm to the Energy Balance Equation,” *IEEE Transactions on Electron Devices*, vol. ED-31, no. 12, pp. 1912–1914, 1984.
- [3] C. C. McAndrew, K. Singhal, and E. L. Heasell, “A Consistent Nonisothermal Extension of the Scharfetter–Gummel Stable Difference Approximation,” *IEEE Electron Device Letters*, vol. EDL-6, no. 9, pp. 446–447, 1985.
- [4] B. Meinerzhagen *et al.*, “A New Highly Efficient Nonlinear Relaxation Scheme for Hydrodynamic MOS Simulations,” in *Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits (NUPAD IV)*, Seattle, WA, USA, pp. 91–96, May 1992.
- [5] 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.
- [6] P. Kaczmarski and P. E. Lagasse, “Bidirectional Beam Propagation Method,” *Electronics Letters*, vol. 24, no. 11, pp. 675–676, 1988.
- [7] K. Rajkanan, R. Singh, and J. Shewchun, “Absorption Coefficient of Silicon for Solar Cell Calculations,” *Solid-State Electronics*, vol. 22, no. 9, pp. 793–795, 1979.
- [8] C. A. Hougen, “Model for infrared absorption and transmission of liquid-phase epitaxy HgCdTe,” *Journal of Applied Physics*, vol. 66, no. 8, pp. 3763–3766, 1989.
- [9] 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.

CHAPTER 15 Radiation Models

This chapter presents 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.

Generation by Gamma Radiation

Using Gamma Radiation Model

The radiation model is activated by specifying the keyword `Radiation(...)` (with optional parameters) in the `Physics` section:

```
Radiation{  
    Dose = <float> | DoseRate = <float>  
    DoseTime = (<float>,<float>)  
    DoseTSigma = <float>  
}
```

where `DoseRate` (in rad/s) represents D in Eq. 474. 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`, `Dose` (in rad) can be specified 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.

15: Radiation Models

Alpha Particles

Yield Function

Generation of electron–hole pairs due to radiation is an electric field–dependent process [1] and is modeled as follows:

$$G_r = g_0 D \cdot Y(F)$$
$$Y(F) = \left(\frac{F + E_0}{F + E_1} \right)^m \quad (474)$$

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 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]  
}
```

Alpha Particles

Using Alpha Particle Model

Specify the `AlphaParticle` in the `Physics` section:

```
Physics { ...  
    AlphaParticle (<optional keywords>)  
}
```

[Table 200 on page 1163](#) lists the keyword options for alpha particles.

Table 81 Coefficients for carrier generation by alpha particles

Symbol	Parameter name	Default value	Unit
s	s	2×10^{-12}	s
w_t	wt	1×10^{-5}	cm
c_2	c2	1.4	1
a	alpha	90	cm^{-1}

Table 81 Coefficients for carrier generation by alpha particles

Symbol	Parameter name	Default value	Unit
α_2	alpha2	5.5×10^{-4}	cm
α_3	alpha3	2×10^{-4}	cm
E_p	Ep	3.6	eV
a_o	a0	-1.033×10^{-4}	cm
a_1	a1	2.7×10^{-10}	cm/eV
a_2	a2	4.33×10^{-17}	cm/eV ²

To plot the instant generation rate $G(t)$ due to alpha particles, specify `AlphaGeneration` in the `Plot` section; to plot $\int_{-\infty}^{\infty} dt G$, specify `AlphaCharge`.

The generation by alpha particles cannot be used except in transient simulations. The amount 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 [Improved Alpha Particle/Heavy Ion Generation Rate Integration on page 497](#).

Alpha Particle Model

The generation rate caused by an alpha particle with energy E is computed by:

$$G(u, v, w, t) = \frac{a}{\sqrt{2\pi \cdot s}} \exp\left[-\frac{1}{2}\left(\frac{t - t_m}{s}\right)^2 - \frac{1}{2}\left(\frac{v^2 + w^2}{w_t^2}\right)\right] \left[c_1 e^{au} + c_2 \exp\left(-\frac{1}{2}\left(\frac{u - \alpha_1}{\alpha_2}\right)^2\right) \right] \quad (475)$$

if $u < \alpha_1 + \alpha_3$, and by:

$$G(u, v, w, t) = 0 \quad (476)$$

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 input file with the keywords `Direction` and `Location` (or `StartPoint`, see [Note on page 495](#)), respectively. Parameter t_m 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, α_1 , is fitted to data [2] by a polynomial function:

$$\alpha_1 = a_0 + a_1 E + a_2 E^2 \quad (477)$$

15: Radiation Models

Heavy Ions

The parameter c_1 is given by:

$$c_1 = \exp[\alpha(\alpha_1[10\text{MeV}] - \alpha_1[E])] \quad (478)$$

The scaling factor a is determined from:

$$\int_0^\infty \int_{-\infty}^\infty \int_{-\infty}^\infty \int_{-\infty}^\infty G(u, v, w, t) dt dw dv du = \frac{E}{E_p} \quad (479)$$

where E_p is the average energy needed to create an electron–hole pair. The remaining parameters are listed in [Table 81 on page 490](#). They are available in the parameter set `AlphaParticle` and are valid for alpha particles with energies between 1 MeV and 10 MeV.

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 may 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.

Using Heavy Ion Model

The simulation of an SEU caused by a heavy ion impact is activated by using the keyword `HeavyIon` in an appropriate `Physics` section:

```
# Default heavy ion
Physics {
    HeavyIon (<keyword_options>) }

# User-defined heavy ion.
Physics {
    HeavyIon (<IonName>) (<keyword_options>) }
# User-defined heavy ions do not have default parameters. Therefore,
# in the parameter file, a specification of the following form must appear:
# HeavyIon(<IonName>){ ... } (see Table 83 on page 495)
```

[Table 201 on page 1163](#) 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 plotted using `HeavyIonCharge` 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, the keyword `LET_f` can be ignored.

An option to improve the spatial integration of the charge generation is presented in [Improved Alpha Particle/Heavy Ion Generation Rate Integration on page 497](#).

Heavy Ion Model

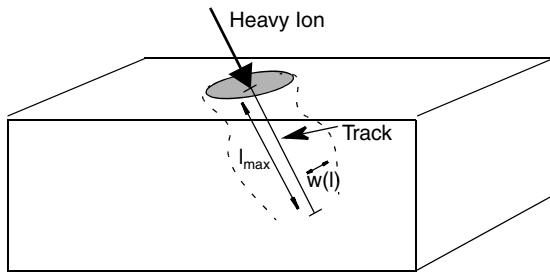


Figure 50 A heavy ion penetrating a semiconductor; its track is defined by a length and the transverse spatial influence is assumed to be symmetric about the track axis

A simple model for the heavy ion impinging process is shown in [Figure 50](#). The generation rate caused by the heavy ion is computed by:

$$G(l, w, t) = G_{\text{LET}}(l)R(w, l)T(t) \quad (480)$$

if $l < l_{\max}$ (l_{\max} is the length of the track), and by:

$$G(l, w, t) = 0 \quad (481)$$

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 linear energy transfer generation density and its unit is pairs/cm³.

$T(t)$ is defined as a Gaussian function:

$$T(t) = \frac{2 \cdot \exp\left(-\left(\frac{t-t_0}{\sqrt{2} \cdot s_{\text{hi}}}\right)^2\right)}{\sqrt{2} \cdot s_{\text{hi}} \sqrt{\pi} \left(1 + \operatorname{erf}\left(\frac{t_0}{\sqrt{2} \cdot s_{\text{hi}}}\right)\right)} \quad (482)$$

15: Radiation Models

Heavy Ions

where t_0 is the moment of the heavy ion penetration (see the keyword `Time` in [Table 201 on page 1163](#)), and s_{hi} is the characteristic value of the Gaussian (see `s_hi` in [Table 83 on page 495](#)).

The spatial distribution, $R(w, l)$, can be defined as an exponential function (default):

$$R(w, l) = \exp\left(-\frac{w}{w_t(l)}\right) \quad (483)$$

or a Gaussian function:

$$R(w, l) = \exp\left(-\left(\frac{w}{w_t(l)}\right)^2\right) \quad (484)$$

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 201 on page 1163](#)).

In addition, the spatial distribution $R(w, l)$ can be defined as a PMI function (see [Spatial Distribution Function on page 1039](#)).

The linear energy transfer (LET) generation density, $G_{\text{LET}}(l)$, is given by:

$$G_{\text{LET}}(l) = a_1 + a_2 l + a_3 e^{a_4 l} + k' \left[c_1 (c_2 + c_3 l)^{c_4} + \text{LET_f}(l) \right] \quad (485)$$

where `LET_f(l)` (defined likewise by the keyword `LET_f`) is a function of the length l . Example 2 in [Examples: Heavy Ions on page 495](#) shows how you can 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/cm³ (default) or pC/μm (activated by the keyword `PicoCoulomb`). Depending on the units of `LET_f` chosen, k' takes on different values in order to make the above equation dimensionally consistent. The appropriate values of k' for different device dimensions are summarized in [Table 82](#). Great care must also be

Table 82 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 ³ for $R(w, l)$ is exponential or Gaussian	$k' = k$	$k' = k$
<code>LET_f</code> has units of pC/μm and $R(w, l)$ is exponential	$k' = \frac{k}{2w_t d}$ $d = 1 \mu\text{m}$	$k' = \frac{k}{2\pi w_t^2}$

Table 82 Setting correct k' to make LET generation density equation dimensionally consistent

Condition	Two-dimensional device	Three-dimensional device
LET_f has units of pC/μm and R(w, l) is Gaussian	$k' = \frac{k}{\sqrt{\pi} w_t d}$ $d = 1 \mu\text{m}$	$k' = \frac{k}{\pi w_t^2}$

Great care must also be exercised to choose the correct units for `Wt_hi` and `Length`. The default unit of `Wt_hi` and `Length` is centimeter. If the keyword `PicoCoulomb` is specified, the unit becomes μm . Examples illustrating the correct unit use are shown in [Examples: Heavy Ions](#). The other coefficients used in Eq. 485 are listed in [Table 83](#) with their default values, and they can be adjusted in the parameter file of Sentaurus Device.

Table 83 Coefficients for carrier generation by heavy ion (Heavylon parameter set)

	<code>s_hi</code>	<code>a₁</code>	<code>a₂</code>	<code>a₃</code>	<code>a₄</code>	<code>k</code>	<code>c₁</code>	<code>c₂</code>	<code>c₃</code>	<code>c₄</code>
Keyword	<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>
Default value	2e-12	0	0	0	0	1	0	1	0	1
Default unit	s	pairs/cm ³	pairs/cm ³ /cm	pairs/cm ³	cm ⁻¹	1	pairs/cm ³	1	cm ⁻¹	1
Unit if PicoCoulomb is chosen	s	pairs/cm ³	pairs/cm ³ /μm	pairs/cm ³	μm ⁻¹	1	pC/μm	1	μm ⁻¹	1

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.

Examples: Heavy Ions

Example 1

The track has a constant `LET_f` value of 0.2 pC/μm across the track. The track length is 1 μm ($l_{\max} = 1 \mu\text{m}$) and the heavy ion crosses the device at the time 0.1 ps. The unit of `LET_f` is pC/μm and the spatial distribution is Gaussian. Since `PicoCoulomb` was chosen, the values of `Length` and `Wt_hi` are expressed in terms of μm.

15: Radiation Models

Heavy Ions

The keyword `HeavyIonChargeDensity` in the `Plot` statement plots the charge density generated by the ion:

```
Physics { Recombination ( SRH(DopingDependence) )
    Mobility (DopingDependence Enormal HighFieldSaturation)
    HeavyIon (
        Direction=(0,1)
        Location=(1.5,0)
        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 linearly interpolated. The unit of `LET_f` is pairs/cm³ (because the keyword `PicoCoulomb` is not used), 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 SEU 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")(
# User-defined heavy ions do not have default parameters. Therefore,
# in the parameter file, a specification of the following form must appear:
# HeavyIon("Ion1"){ ... } (see Table 83 on page 495)
  Direction=(0,1)
  Location=(1,0)
  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] )
}

```

Improved Alpha Particle/Heavy Ion Generation Rate Integration

Accurate integration of alpha particle or heavy ion generation rates is very important for predictive modeling of SEU phenomena. By default, in Sentaurus Device, the integration of the generation rate over the control volume associated with each vertex in the mesh is performed under the assumption that the generation rate is constant inside the vertex control volume and equal to the generation rate value at the vertex. As alpha particle or heavy ion generation rates can change very rapidly in space, the approximation error with such an approach may lead to large errors on a coarse mesh (in particular, the method does not guarantee charge conservation).

To eliminate this source of numeric error, an improved spatial integration can be performed. The procedure used in Sentaurus Device for optical generation (see [Chapter 14 on page 393](#)) extends to the alpha particle/heavy ion case. Each control volume is covered by a set of small rectangular boxes and the generation rate is integrated numerically inside these boxes. To activate this procedure, the keyword `RecBoxIntegr` must be specified in the `Math` section. The same keyword is used as for optical generation, with the same set of default parameters. By changing the default parameters, you can also control the accuracy of the integration (see [Optical Beam Absorption on page 434](#)).

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] 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.

CHAPTER 16 Noise and Fluctuation Analysis

This chapter discusses noise and fluctuation analysis in Sentaurus Device.

Noise analysis is concerned with deviations from the average behavior that occur dynamically in an average device. Fluctuation analysis is concerned with (static) deviations of device properties from an average device. [Performing Noise and Fluctuation Analysis](#) explains how to perform noise and fluctuation analysis. Deviations occur for a multitude of reasons; for each of them, a physical model is required, and the availability of models determines the kind of noise or fluctuation that can be modeled. [Noise Sources on page 501](#) discusses the models that Sentaurus Device offers. [Impedance Field Method on page 504](#) discusses the background of the impedance field method [1], and [Noise Output Data on page 506](#) summarizes the data available for visualization.

Performing Noise and Fluctuation Analysis

Sentaurus Device models noise and fluctuations of device properties in a unified manner as an extension of small-signal analysis (see [ACCoupled: Small-Signal AC Analysis on page 161](#)).

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.

To use noise and fluctuation analysis, first 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 ( <Noisemodels> )  
}
```

where `<Noisemodels>` is a list that specifies any number of noise models listed in [Table 216 on page 1172](#). As for other physical models, noise sources can be specified regionwise or materialwise.

16: Noise and Fluctuation Analysis

Performing Noise and Fluctuation Analysis

Second, use the `ObservationNode` option to the `ACCoupled` statement to specify the device nodes for which the noise voltage spectral densities are desired, 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 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 506](#)). For more information on the `ACCoupled` statement, see [ACCoupled: Small-Signal AC Analysis on page 161](#).

NOTE The observation nodes must be a subset of the nodes specified in `Node(...)`.

The results of the analysis are the noise voltage spectral densities and the noise current spectral densities. They appear in the `ACEExtract` file (see [Table 84](#)). The units are V^2 's for the voltage spectral densities and A^2 's for the current noise spectral densities.

Table 84 Noise voltage spectral densities

Name	Description
S_V	Autocorrelation noise voltage spectral density (NVSD)
S_I	Autocorrelation noise current spectral density (NISD)
S_V_ee S_V_hh	Electron NVSD Hole NVSD
S_V_eeDiff S_V_hhDiff	Electron NVSD due to diffusion LNS Hole NVSD due to diffusion LNS
S_V_eeMonoGR S_V_hhMonoGR	Electron NVSD due to monopolar GR LNS Hole NVSD due to monopolar GR LNS
S_V_eeFlickerGR S_V_hhFlickerGR	Electron NVSD due to flicker GR LNS Hole NVSD due to flicker GR LNS
S_V_Doping	NVSD due to random dopant fluctuations
S_V_Trap	NVSD due to trapping noise

Table 84 Noise voltage spectral densities

Name	Description
ReS_VXV ImS_VXV	Real/imaginary parts of the cross correlation noise voltage spectral density (NVXVSD)
ReS_IXI ImS_IXI	Real/imaginary parts of the cross correlation noise current spectral density
ReS_VXV_ee ImS_VXV_ee ReS_VXV_hh ImS_VXV_hh	Real/imaginary parts of the electron/hole NVXVSD
ReS_VXV_eeDiff ImS_VXV_eeDiff ReS_VXV_hhDiff ImS_VXV_hhDiff	Real/imaginary parts of the electron/hole NVXVSD due to diffusion LNS
ReS_VXV_eeMonogr ImS_VXV_eeMonogr ReS_VXV_hhMonogr ImS_VXV_hhMonogr	Real/imaginary parts of the electron/hole NVXVSD due to monopolar GR LNS
ReS_VXV_eeFlickerGR ImS_VXV_eeFlickerGR ReS_VXV_hhFlickerGR ImS_VXV_hhFlickerGR	Real/imaginary parts of the electron/hole NVXVSD due to flicker GR LNS
ReS_VXV_Doping ImS_VXV_Doping	Real/imaginary parts of the NVXVSD due to random dopant fluctuations
ReS_VXV_Trap ImS_VXV_Trap	Real/imaginary parts of the NVXVSD due to trapping noise

Noise Sources

Diffusion Noise

The diffusion noise source (keyword `DiffusionNoise`) available in Sentaurus Device reads:

$$K_{n,n}^{\text{Diff}}(\vec{r}_1, \vec{r}_2, \omega) = 4qn(\vec{r}_1)kT_n(\vec{r}_1)\mu_n(\vec{r}_1)\delta(\vec{r}_1 - \vec{r}_2) \quad (486)$$

A corresponding expression is used for holes. $K_{n,n}^{\text{Diff}}$ is a diagonal tensor.

16: Noise and Fluctuation Analysis

Noise Sources

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`, `eTemperature`, `hTemperature`, or `e_h_Temperature`. The default is `LatticeTemperature`.

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]:

$$K_{n,n}^{\text{GR}}(\vec{r}_1, \vec{r}_2, \omega) = \frac{\vec{J}_n(\vec{r}_1)\vec{J}(\vec{r}_1)}{n} \cdot \frac{4\alpha\tau_{\text{eq}}}{1 + \omega^2\tau_{\text{eq}}^2} \delta(\vec{r}_1 - \vec{r}_2) \quad (487)$$

where α is a fitting parameter and τ_{eq} an equivalent GR lifetime. The parameters τ_{eq} and α can be modified in the parameter file of Sentaurus Device. A similar expression is used for holes.

Bulk Flicker Noise

The flicker generation–recombination (GR) noise model (keyword `FlickerGRNoise`) for electrons (similar for holes) is:

$$K_{n,n}^{\text{fGR}}(\vec{r}_1, \vec{r}_2, \omega) = \frac{\vec{J}_n(\vec{r}_1)\vec{J}(\vec{r}_1)}{n(\vec{r}_1)} \frac{2\alpha_H}{\pi v \ln(\tau_1/\tau_0)} [\text{atan}(\omega\tau_1) - \text{atan}(\omega\tau_0)] \delta(\vec{r}_1 - \vec{r}_2) \quad (488)$$

where α_H is a fit parameter; $\omega = 2\pi\nu$, the angular frequency; and the time constants fulfill $\tau_0 < \tau_1$. The parameters α_H , τ_0 , and τ_1 for electrons and holes are accessible in the parameter

file. With increasing frequency, the noise source changes from constant to $1/v$ behavior close to the frequency $v_1 = 1/\tau_1$ and, ultimately, to $1/v^2$ behavior at $v_0 = 1/\tau_0$.

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. All trap models apart from tunneling to electrodes are supported. For more details on traps, see [Chapter 10 on page 349](#).

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 :

$$R_{i,k} = N_{t,k}(1-f_k)c_{i,k} - N_{t,k}f_k e_{i,k} + s_{i,k} \quad (489)$$

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 [Eq. 367, p. 352](#)). Denoting the expectation value with $\langle \rangle$, the trapping noise source takes the form:

$$K_{ij,k}^{\text{trap}}(\omega) = 2q^2 \int dt \langle s_{i,k}(0)s_{j,k}(t) \rangle \exp(-i\omega t) = 2q^2 \delta_{ij} N_{t,k} [(1-f_k)c_{i,k} + f_k e_{i,k}] \quad (490)$$

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:

$$K_A^{\text{RDF}}(\vec{r}_1, \vec{r}_2, \omega) = N_A(\vec{r}_1) \frac{\Theta(0.5\text{Hz} - |v|)}{1\text{Hz}} \delta(\vec{r}_1 - \vec{r}_2) \quad (491)$$

Here, $v = \omega/2\pi$ is the frequency. An analogous expression holds for the noise source for donors, K_D^{RDF} . Physically, the noise sources are static. However, to avoid a δ -function in frequency space, Sentaurus Device spreads the spectral density of the noise source over a 1 Hz frequency interval. [Eq. 491](#) is based on the assumption that individual dopant atoms are completely uncorrelated.

16: Noise and Fluctuation Analysis

Impedance Field Method

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.

Noise from SPICE Circuit Elements

To take into account the noise generated by SPICE circuit elements (see [Compact Models User Guide, Chapter 1 on page 1](#)), specify the `CircuitNoise` option to `ACCCoupled`. 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 circuit elements as noiseless. Non-SPICE compact circuit elements do not implement noise at all and, therefore, Sentaurus Device always treats them as noiseless.

Impedance Field Method

The impedance field method splits noise and fluctuation analysis into two tasks. The first task is to provide models for the noise sources, that is, for the local microscopic fluctuations inside the devices (see [Noise Sources on page 501](#)). The selection of the appropriate models depends on the problem. You have to pick the models according to the kind of noise they are interested in. 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 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 $G_\xi^Y(x, x', \omega)$ of the potential at location x due to a perturbation at location x' with angular frequency ω in the right-hand side of the partial differential equation for solution variable ξ . ξ can be ϕ (see [Eq. 26, p. 178](#)), n or p (see [Eq. 27, p. 178](#)), T_n (see [Eq. 38, p. 185](#)), T_p (see [Eq. 39, p. 185](#)) or T .

The implemented models result in the following expression for the noise voltage spectral density:

$$S_V(x, x', \omega) = \int \underline{G}_n^V(x, x'', \omega) K_{n,n}^{\text{Diff}}(x'', \omega) \underline{G}_n^{V*}(x', x'', \omega) dx'' \quad (492)$$

$$+ \int \underline{G}_n^V(x, x'', \omega) K_{n,n}^{\text{GR}}(x'', \omega) \underline{G}_n^{V*}(x', x'', \omega) dx'' \quad (493)$$

$$+ \int \underline{G}_n^V(x, x'', \omega) K_{n,n}^{\text{fGR}}(x'', \omega) \underline{G}_n^{V*}(x', x'', \omega) dx'' \quad (494)$$

$$+ \int \underline{G}_p^V(x, x'', \omega) K_{p,p}^{\text{Diff}}(x'', \omega) \underline{G}_p^{V*}(x', x'', \omega) dx'' \quad (495)$$

$$+ \int \underline{G}_p^V(x, x'', \omega) K_{p,p}^{\text{GR}}(x'', \omega) \underline{G}_p^{V*}(x', x'', \omega) dx'' \quad (496)$$

$$+ \int \underline{G}_p^V(x, x'', \omega) K_{p,p}^{\text{fGR}}(x'', \omega) \underline{G}_p^{V*}(x', x'', \omega) dx'' \quad (497)$$

$$+ \int G_A^V(x, x'', \omega) K_A^{\text{RDF}}(x'', \omega) G_A^{V*}(x', x'', \omega) dx'' \quad (498)$$

$$+ \int G_D^V(x, x'', \omega) K_D^{\text{RDF}}(x'', \omega) G_D^{V*}(x', x'', \omega) dx'' \quad (499)$$

$$+ \int \sum_{ik} G_{\text{trap}, i,k}^V(x, x'', \omega) K_{ii,k}^{\text{trap}}(x'', \omega) G_{\text{trap}, i,k}^{V*}(x', x'', \omega) dx'' \quad (500)$$

where the K are the noise sources (see [Noise Sources on page 501](#)). The spatial coordinates x and x' each correspond to a contact.

For drift-diffusion simulations, $\underline{G}_n^V = \nabla G_n^V$. For hydrodynamic simulations, \underline{G}_n^V is replaced by an effective Green's function, $\underline{G}_n^V = \nabla G_n^V + G_{T_n}^V \nabla E_C - 5r_n k T_n \nabla G_{T_n}^V / 2$ (see [Hydrodynamic Transport Model on page 184](#)). Similar relations hold for \underline{G}_p^V .

The Green's function G_A^V for random dopant fluctuations reads:

$$G_A^V = G_n^V \frac{\partial \nabla \cdot \vec{J}_n}{\partial N_A} + G_p^V \frac{\partial \nabla \cdot \vec{J}_p}{\partial N_A} + G_{T_n}^V \frac{\partial (\nabla \cdot \vec{S}_n - \vec{J}_n \cdot \nabla E_C)}{\partial N_A} + G_{T_p}^V \frac{\partial (\nabla \cdot \vec{S}_p - \vec{J}_p \cdot \nabla E_V)}{\partial N_A} - G_\phi^V \quad (501)$$

An analogous expression holds for G_D^V . The expression above is valid for hydrodynamic simulations. The expression for drift-diffusion simulations is similar. The derivatives with respect to N_A describe the impact of dopant fluctuations on mobility and bandgap narrowing. Sentaurus Device takes them into account only if explicitly requested by you (see [Random Dopant Fluctuations on page 503](#)).

16: Noise and Fluctuation Analysis

Noise Output Data

The Green's function $G_{\text{trap}, i, k}^V$ for trap level k and the capture or emission process i is not given here in detail. Interested readers are referred to the literature [3], where full expressions for an equivalent approach are presented.

The equations for the noise current spectral densities are obtained from those for the noise voltage spectral densities by replacing the Green's functions G_ξ^V by the Green's functions G_ξ^I that denote the responses of node currents to fluctuations inside the device.

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 85 on page 507](#). In the tables, the abbreviations LNS (local noise source) and LNVSD (local noise voltage spectral density) are used.

Autocorrelation data refers to [Eq. 492](#) to [Eq. 498](#), when x and x' are identical. 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 `ACCoupled`.

In the case of $x \neq x'$ in [Eq. 492](#) to [Eq. 498](#), 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
```

Table 85 Device noise data

Keyword	Description	Reference equation
eeDiffusionLNS	Electron diffusion LNS	Eq. 486
hhDiffusionLNS	Hole diffusion LNS	
eeMonopolarGRLNS	Trace of electron monopolar GR LNS	Eq. 487
hhMonopolarGRLNS	Trace of hole monopolar GR LNS	
eeFlickerGRLNS	Trace of electron flicker GR LNS	Eq. 488
hhFlickerGRLNS	Trace of hole flicker GR LNS	
ReLNVXVSD ImLNVXVSD	Real/imaginary parts of LNVSD	Sum of integrands in Eq. 492–Eq. 500
ReLNISD ImLNISD	Real/imaginary parts of local noise current spectral density	Sum of integrands in Eq. 492–Eq. 500 with G_{ξ}^V replaced by G_{ξ}^I
ReeeLNVXVSD ImeeLNVXVSD	Real/imaginary parts of LNVSD for electrons	Sum of integrands in Eq. 492–Eq. 494
RehhLNVXVSD ImhhLNVXVSD	Real/imaginary parts of LNVSD for holes	Sum of integrands in Eq. 495–Eq. 497
ReeeDiffusionLNVXVSD ImeeDiffusionLNVXVSD	Real/imaginary parts of diffusion LNVSD for electrons	Integrand of Eq. 492
RehhDiffusionLNVXVSD ImhhDiffusionLNVXVSD	Real/imaginary parts of diffusion LNVSD for holes	Integrand of Eq. 495
ReeeMonopolarGRLNVXVSD ImeeMonopolarGRLNVXVSD	Real/imaginary parts of electron monopolar LNVSD	Integrand of Eq. 493
RehhMonopolarGRLNVXVSD ImhhMonopolarGRLNVXVSD	Real/imaginary parts of hole monopolar LNVSD	Integrand of Eq. 496
ReeeFlickerGRLNVXVSD ImeeFlickerGRLNVXVSD	Real/imaginary parts of electron flicker GR LNVSD	Integrand of Eq. 494
RehhFlickerGRLNVXVSD ImhhFlickerGRLNVXVSD	Real/imaginary parts of hole flicker GR LNVSD	Integrand of Eq. 497
ReTrapLNISD ImTrapLNISD	Real/imaginary parts of local trapping noise current spectral density	Integrand of Eq. 500 with G_{ξ}^V replaced by G_{ξ}^I
ReTrapLNVSD ImTrapLNVSD	Real/imaginary parts of local trapping noise voltage spectral density	Integrand of Eq. 500
PoECReACGreenFunction PoECImACGreenFunction	Real/imaginary parts of G_n^V	
PoHCRReACGreenFunction PoHCImACGreenFunction	Real/imaginary parts of G_p^V	

16: Noise and Fluctuation Analysis

Noise Output Data

Table 85 Device noise data

Keyword	Description	Reference equation
PoETReACGreenFunction PoETImACGreenFunction	Real/imaginary parts of $G_{T_n}^V$	
PoHTReACGreenFunction PoHTImACGreenFunction	Real/imaginary parts of $G_{T_p}^V$	
PoLTReACGreenFunction PoLTImACGreenFunction	Real/imaginary parts of G_T^V	
PoPotReACGreenFunction PoPotImACGreenFunction	Real/imaginary parts of G_ϕ^V	
CurECReACGreenFunction CurECImACGreenFunction CurHCReACGreenFunction CurHClmACGreenFunction CurETReACGreenFunction CurETImACGreenFunction CurHTReACGreenFunction CurHTImACGreenFunction CurLTReACGreenFunction CurLTImACGreenFunction CurPotReACGreenFunction CurPotImACGreenFunction	Real/imaginary parts of $G_n^I, G_p^I, G_{T_n}^I, G_{T_p}^I, G_T^I$, and G_ϕ^I	
GradPoECReACGreenFunction GradPoECImACGreenFunction GradPoHCReACGreenFunction GradPoHClmACGreenFunction GradPoETReACGreenFunction GradPoETImACGreenFunction GradPoHTReACGreenFunction GradPoHTImACGreenFunction	Real/imaginary parts of $\nabla \cdot G_n^V, \nabla \cdot G_p^V, \nabla \cdot G_{T_n}^V, \nabla \cdot G_{T_p}^V$	
Grad2PoECACGreenFunction Grad2PoHCACGreenFunction	$ G_n^V ^2$ and $ G_p^V ^2$	
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.

16: Noise and Fluctuation Analysis

References

CHAPTER 17 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 9 on page 309](#)). 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 18 on page 531](#). Tunneling to traps is discussed in [Tunneling and Traps on page 357](#).

Tunneling Model Overview

Sentaurus Device offers three tunneling models. The most versatile tunneling model is the nonlocal tunneling model (see [Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 518](#)). This model:

- 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.

The second most powerful model is the direct tunneling model (see [Direct Tunneling on page 514](#)). This model:

- 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).

17: Tunneling

Fowler–Nordheim Tunneling

Use this model to describe leakage through thin gate insulators, provided those are of uniform or of uniformly graded composition.

The simplest tunneling model is the Fowler–Nordheim model (see [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 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.

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 2D, 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 Gates on page 211](#)). The additional floating gate capacitance can be specified as FGcap in the Electrode statement (see [Electrode Section on page 50](#)).

Fowler–Nordheim Tunneling

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 513](#).

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.

The Fowler–Nordheim tunneling model can be used with any of the hot-carrier injection models (see [Chapter 18 on page 531](#)), for example:

```
GateCurrent( Fowler Lucky )
```

switches on the Fowler–Nordheim tunneling model and the classical lucky electron injection model simultaneously.

Fowler–Nordheim Model

The Fowler–Nordheim model reads:

$$j_{FN} = AF_{ins}^2 \exp\left(-\frac{B}{F_{ins}}\right) \quad (502)$$

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 Tecplot SV 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 $\gamma > 1$:

$$j_n = \gamma \cdot j_{FN} \quad (503)$$

$$j_p = (\gamma - 1) \cdot j_{FN} \quad (504)$$

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 input 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. 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 545](#)).

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 86](#) lists the parameters and their default values.

Table 86 Coefficients for Fowler–Nordheim tunneling (defaults for silicon–oxide interface)

Symbol	Parameter name	Default value	Unit	Remarks
A (erase)	<code>Ae</code>	1.87×10^{-7}	A/V^2	A for the erase cycle
B (erase)	<code>Be</code>	1.88×10^8	V/cm	B for the erase cycle
A (write)	<code>Aw</code>	1.23×10^{-6}	A/V^2	A for the write cycle
B (write)	<code>Bw</code>	2.37×10^8	V/cm	B for the write cycle
γ	<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.

Using Direct Tunneling

The direct tunneling model is specified as an option of the `GateCurrent` statement (see [Using Fowler–Nordheim on page 512](#)) 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 18 on page 531](#)) are as for the Fowler–Nordheim model, see [Fowler–Nordheim Tunneling on page 512](#).

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 517](#).

To plot the direct tunneling current, the keywords `eDirectTunneling` or `hDirectTunneling` must be included in the `Plot` section:

```
Plot {eDirectTunneling hDirectTunneling}
```

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:

$$j_n = \frac{qm_C k}{2\pi^2 \hbar^3} \int_0^\infty dE \Upsilon(E) \left\{ T(0) \ln \left(\exp \left[\frac{E_{F,n}(0) - E_C(0) - E}{kT(0)} \right] + 1 \right) \right. \\ \left. - T(d) \ln \left(\exp \left[\frac{E_{F,n}(d) - E_C(0) - E}{kT(d)} \right] + 1 \right) \right\} \quad (505)$$

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 $E_C(0)$).

$\Upsilon(E) = 2/(1+g(E))$ is the transmission coefficient for a trapezoidal potential barrier, with:

$$g(E) = \frac{\pi^2}{2} \left\{ \sqrt{\frac{E_T}{E}} \sqrt{\frac{m_{Si}}{m_G}} (Bi'_d Ai_0 - Ai'_d Bi_0)^2 + \sqrt{\frac{E}{E_T}} \sqrt{\frac{m_G}{m_{Si}}} (Bi_d Ai'_0 - Ai_d Bi'_0)^2 + \right. \\ \left. \frac{\sqrt{m_G m_{Si}} \hbar \Theta_{ox}}{m_{ox} \sqrt{EE_T}} (Bi'_d Ai'_0 - Ai'_d Bi'_0)^2 + \frac{m_{ox}}{\sqrt{m_G m_{Si}}} \frac{\sqrt{EE_T}}{\hbar \Theta_{ox}} (Bi_d Ai_0 - Ai_d Bi_0) \right\} \quad (506)$$

and:

$$Ai_0 = Ai\left(\frac{E_B(E) - E}{\hbar \Theta_{ox}}\right), \quad Ai_d = Ai\left(\frac{E_B(E) - qF_{ox}d - E}{\hbar \Theta_{ox}}\right) \quad (507)$$

and so on, where $\hbar\Theta_{\text{ox}} = (q^2\hbar^2F_{\text{ox}}^2/2m_{\text{ox}})^{1/3}$ 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. $F_{\text{ox}} = V_{\text{ox}}/d$ 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. A_i and B_i are Airy functions, and A'_i and B'_i 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:

$$E_B(E) = E_B(E_0) + \frac{E_B(E_2) - E_B(E_0)}{(E_2 - E_0)(E_1 - E_2)}(E - E_0)(E_1 - E) - \frac{E_B(E_1) - E_B(E_0)}{(E_1 - E_0)(E_1 - E_2)}(E - E_0)(E_2 - E) \quad (508)$$

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:

$$S_{\text{tra}}(E) = S_{\text{im}}(E) \quad (509)$$

is solved for $E_j(j = 0, 1, 2)$, which results in three pseudobarrier heights $E_B(E_j)$ used in Eq. 508. In Eq. 509, $S_{\text{tra}}(E)$ is the action of the trapezoidal pseudobarrier:

$$S_{\text{tra}}(E) = \frac{2}{3} \left[\left| \frac{E_B(E) - qF_{\text{ox}}d - E}{\hbar\Theta_{\text{ox}}} \right|^{\frac{3}{2}} \Theta[E_B(E) - qF_{\text{ox}}d - E] - \left| \frac{E_B(E) - E}{\hbar\Theta_{\text{ox}}} \right|^{\frac{3}{2}} \right] \quad (510)$$

and $S_{\text{im}}(E)$ is the action of the respective image potential barrier:

$$S_{\text{im}}(E) = \sqrt{\frac{2m_{\text{ox}}}{\hbar^2}} \int_{x_l(E)}^{x_r(E)} d\xi \sqrt{E_B - qF_{\text{ox}}\xi + E_{\text{im}}(\xi) - E} \quad (511)$$

$x_{l,r}(E)$ denotes the classical turning points for a given carrier energy that follow from:

$$E_T - qF_{\text{ox}}x_{l,r} + E_{\text{im}}(x_{l,r}) = E \quad (512)$$

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:

$$E_{\text{im}}(x) = \frac{q^2}{16\pi\epsilon_{\text{ox}}} \sum_{n=0}^{10} (k_1 k_2)^n \left[\frac{k_1}{nd+x} + \frac{k_2}{d(n+1)-x} + \frac{2k_1 k_2}{d(n+1)} \right] \quad (513)$$

with:

$$k_1 = \frac{\epsilon_{\text{ox}} - \epsilon_{\text{Si}}}{\epsilon_{\text{ox}} + \epsilon_{\text{Si}}}, \quad k_2 = \frac{\epsilon_{\text{ox}} - \epsilon_{\text{G}}}{\epsilon_{\text{ox}} + \epsilon_{\text{G}}} = -1 \quad (514)$$

In Eq. 514, ϵ_{ox} , ϵ_{Si} , and ϵ_{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 87. The parameters are specified according to the interface.

Table 87 Coefficients for direct tunneling (defaults for oxide barrier on silicon)

Symbol	Parameter name	Electrons	Holes	Unit	Description
ϵ_{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_{ox}	<code>m_ins</code>	0.50	0.77	m_0	Effective mass in insulator
m_{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
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

17: Tunneling

Nonlocal Tunneling at Interfaces, Contacts, and Junctions

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 520](#)).
3. Adjust the physical and numeric parameters (see [Nonlocal Tunneling Parameters on page 521](#)).

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. Nonlocal meshes can be constructed using two different forms:

- In the simpler form, `NonLocal` specifies barrier regions. For example:

```
Math { Nonlocal "NLM" ( Barrier(Region="oxtop" Region="oxbottom") ) }
```

constructs a nonlocal mesh for a tunneling barrier that consists of the regions `oxtop` and `oxbottom`. See [Specification Using Barrier on page 107](#) for details.
- 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. For example, with:

```
Math { Nonlocal "NLM" (Electrode="Gate" Length=5e-7) }
```

Sentaurus Device constructs a nonlocal mesh called "NLM" that connects vertices up to a distance of 5 nm to the Gate electrode. See [Specification Using a Reference Surface on page 108](#) for details.

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 [Tunneling and Traps on page 357](#)), 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 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 156 on page 1124](#). For details about the nonlocal mesh and its construction, see [Constructing Nonlocal Meshes on page 107](#).

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 107](#)).

17: Tunneling

Nonlocal Tunneling at Interfaces, Contacts, and Junctions

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 [Eq. 524, p. 527](#). 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 [Eq. 524](#) and [Eq. 526, p. 528](#). To include contributions by tunneling to and from the 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 [Eq. 342, p. 339](#) instead of [Eq. 526](#) to calculate the band-to-band tunneling contribution, which makes the model consistent with the [Dynamic Nonlocal Path Band-to-Band Model on page 339](#) 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 110](#)).

Figure 51 illustrates the four contributions to the total tunneling current.

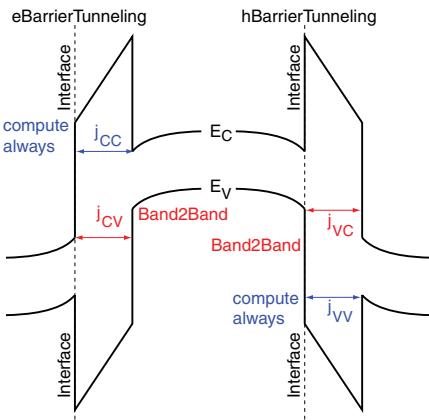


Figure 51 Various nonlocal tunneling current contributions

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 166 on page 1137](#). For a detailed discussion of the physics of the nonlocal tunneling model, see [Physics of Nonlocal Tunneling Model on page 523](#).

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 [Eq. 523](#) and [Eq. 525](#)). 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 [Eq. 515](#) and [Eq. 516](#)). 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 Schrödinger (see [Eq. 520](#), p. 525 and [Schrödinger Equation-based Tunneling Probability on page 526](#)).

17: Tunneling

Nonlocal Tunneling at Interfaces, Contacts, and Junctions

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.

`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 α_n and α_p for the quantization corrections (see [Density Gradient Quantization Correction on page 526](#)). 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).

For example:

```
BarrierTunneling {  
    mt = 0.5, 0.5  
    g = 1, 2  
}  
Material = "Oxide" {  
    BarrierTunneling {  
        mt = 0.42, 1.0  
    }  
}
```

changes the prefactors g_C and g_V to 1 and 2, respectively. 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.

`BarrierTunneling` can also be applied specifically to named nonlocal meshes. For example:

```
Material = "Oxide" {  
    BarrierTunneling "NLM" {  
        mt = 0.42, 1.0  
    }  
}
```

changes the oxide tunneling mass for the nonlocal mesh `NLM` only. Specifications for named nonlocal meshes take precedence over the general specifications.

Specify numeric parameters for the model in the Math section specific to the interface or contact to which the nonlocal tunneling model is applied. The parameter EnergyResolution(NonLocal) (given in eV) is a lower limit for the energy step that Sentaurus Device uses to perform integrations over band-edge energies.

The parameter Digits(Nonlocal) determines the relative accuracy (the number of valid decimal digits) to which Sentaurus Device computes the tunneling currents. The default value for Digits(Nonlocal) is 2, whereas for EnergyResolution(NonLocal), it is 0.005.

For example:

```
Math {
    NonLocal "NLM" (
        ...
        Digits=3
        EnergyResolution=0.001
    )
}
```

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.

Visualizing Nonlocal Tunneling

To visualize nonlocal tunneling, specify the keyword eBarrierTunneling or hBarrierTunneling in the Plot section (see [Plot Section on page 58](#)). 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.

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.

17: 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 ε :

$$\kappa_{C,v}(r, \varepsilon) = \sqrt{2m_C(r)|E_{C,v}(r) - \varepsilon|} \Theta[E_{C,v}(r) - \varepsilon]/\hbar \quad (515)$$

$$\kappa_{V,v}(r, \varepsilon) = \sqrt{2m_V(r)|\varepsilon - E_{V,v}(r)|} \Theta[\varepsilon - E_{V,v}(r)]/\hbar \quad (516)$$

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 521](#)). $E_{C,v}$ 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 521](#)).

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 ε can be written as:

$$\Gamma_{CC,v}(u, l, \varepsilon) = T_{CC,v}(l, \varepsilon) \exp \left[-2 \int_l^u \kappa_{C,v}(r, \varepsilon) dr \right] T_{CC,v}(u, \varepsilon) \quad (517)$$

and:

$$\Gamma_{VV,v}(u, l, \varepsilon) = T_{VV,v}(l, \varepsilon) \exp \left[-2 \int_l^u \kappa_{V,v}(r, \varepsilon) dr \right] T_{VV,v}(u, \varepsilon) \quad (518)$$

If the option `TwoBand` is specified to `eBarrierTunneling` (see [Table 166 on page 1137](#)), Sentaurus Device replaces $\kappa_{C,v}$ in [Eq. 517](#) with the two-band dispersion relation:

$$\kappa_v = \frac{\kappa_{C,v} \kappa_{V,v}}{\sqrt{\kappa_{C,v}^2 + \kappa_{V,v}^2}} \quad (519)$$

If the option `TwoBand` is specified to `hBarrierTunneling`, Sentaurus Device replaces $\kappa_{V,v}$ in [Eq. 518](#) with the two-band relation [Eq. 519](#). Near the conduction and the valence band edge, the two-band dispersion relation approaches the single band dispersion relations [Eq. 515](#) and [Eq. 516](#), and provides a smooth interpolation in between. [Figure 52 on page 525](#) illustrates this.

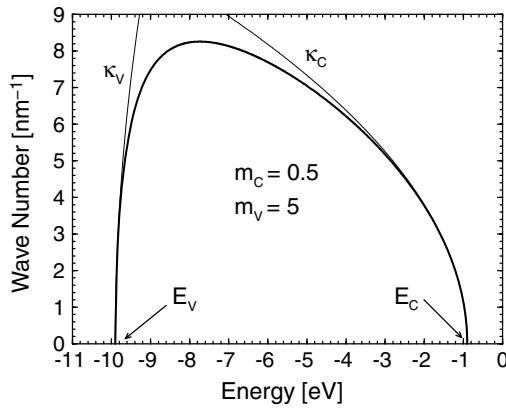


Figure 52 Comparison of two-band and single-band dispersion relations

The two-band dispersion relation does not distinguish between electrons and holes. In particular, for the two-band dispersion relation, $\Gamma_{CC,v} = \Gamma_{VV,v}$.

The two-band dispersion relation is most useful when band-to-band tunneling is active (keyword `Band2Band`, see [Table 166 on page 1137](#)). 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 [Eq. 517](#) and [Eq. 518](#) equal one. If the Transmission option is specified to `eBarrierTunneling` [3]:

$$T_{CC,v}(x, \varepsilon) = \frac{v_{-,v}(x, \varepsilon) \sqrt{v_{-,v}(x, \varepsilon)^2 + 16v_{+,v}(x, \varepsilon)^2}}{v_{+,v}(x, \varepsilon)^2 + v_{-,v}(x, \varepsilon)^2} \quad (520)$$

Here, $v_{-,v}(x, \varepsilon)$ denotes the velocity of a particle with energy ε 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 [Eq. 519](#)). 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 [Eq. 517](#) and [Eq. 518](#), 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 [Eq. 520](#)) 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`.

17: Tunneling

Nonlocal Tunneling at Interfaces, Contacts, and Junctions

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 metal or contact. The masses used to compute the velocities are the tunneling effective masses for the metal or contact.

Schrödinger Equation–based Tunneling Probability

In addition to the WKB-based models discussed in [WKB Tunneling Probability on page 524](#), 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:

$$\left(-\frac{d}{dr}m(r)\frac{d}{dr} + E_{C,v}(r)\right)\Psi(r) = E(r)\Psi(r) \quad (521)$$

between the outermost classical turning points that belong to the tunneling energy E . For $m(r)$, Sentaurus Device uses the tunneling mass m_t (see [Nonlocal Tunneling Parameters on page 521](#)). $E_{C,v}$ is the conduction-band energy shifted by the v -th value in `eoffset` (see [Nonlocal Tunneling Parameters on page 521](#)).

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 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.

Density Gradient Quantization Correction

In combination with the density gradient model (see [Density Gradient Quantization Model on page 260](#)), the nonlocal tunneling model offers an optional correction for quantization effects. The corrections are multipliers to $T_{CC,v}(r, \epsilon)$ and read:

$$\exp\left(\alpha_n \frac{\Lambda_{fb,n}(r) - \Lambda_n(r)}{kT_n(r)}\right) \quad (522)$$

where α_n is an adjustable parameter (see [Nonlocal Tunneling Parameters on page 521](#)), 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 Eq. 522 exists for $T_{VV,v}$.

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 ε is:

$$R_{CC,v}(u, l, \varepsilon) - G_{CC,v}(u, l, \varepsilon) = \frac{A_{CC}}{qk} \vartheta \left[\varepsilon - E_{C,v}(u), -\frac{dE_{C,v}}{du}(u) \right] \vartheta \left[\varepsilon - E_{C,v}(l), \frac{dE_{C,v}}{dl}(l) \right] \Gamma_{CC,v}(u, l, \varepsilon) \times \quad (523)$$

$$\left[T_n(u) \ln \left(1 + \exp \left[\frac{E_{F,n}(u) - \varepsilon}{kT_n(u)} \right] \right) - T_n(l) \ln \left(1 + \exp \left[\frac{E_{F,n}(l) - \varepsilon}{kT_n(l)} \right] \right) \right]$$

where $\vartheta(x, y) = \delta(x)|y|\Theta(y)$, $A_{CC} = g_C A_0$ is the effective Richardson constant (with A_0 the Richardson constant for free electrons), and g_C is a fit parameter (see [Nonlocal Tunneling Parameters on page 521](#)). $R_{CC,v}$ relates to the first term and $G_{CC,v}$ to the second term in the second line of [Eq. 523](#). $\Gamma_{CC,v}$ is the tunneling probability (see [Eq. 517](#)). For nonlocal lines with zero Permeation, the second ϑ -function is replaced with $\Theta[\varepsilon - E_{C,v}(l)]\delta(l - 0^+)$ (with 0^+ infinitesimally smaller than 0). For contacts, metals, or in the presence of the option BandGap, the second ϑ -function is replaced with $\delta(l - 0^+)$.

The current density of 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 ([Eq. 523](#)):

$$j_{CC}(l) = -q \sum_v \int_{-\infty}^{\infty} [R_{CC,v}(u, l, \varepsilon) - G_{CC,v}(u, l, \varepsilon)] d\varepsilon du \quad (524)$$

The options for `hBarrierTunneling` and the expressions for the valence band to valence band tunneling current density j_{VV} are analogous.

Band-to-Band Contributions to Nonlocal Tunneling Current

When you specify the `Band2Band` option to `eBarrierTunneling` (see [Table 166 on page 1137](#)), the total current that flows to the conduction band at point l also contains electrons that originate from the valence band at position $u > l$. The recombination rate of the valence-band electrons with energy ε 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:

$$R_{CV,v}(u, l, \varepsilon) - G_{CV,v}(u, l, \varepsilon) = \frac{A_{CV}}{2qk} \vartheta \left[\varepsilon - E_{V,v}(u), \frac{dE_{V,v}}{du}(u) \right] \vartheta \left[\varepsilon - E_{C,v}(l), \frac{dE_{C,v}}{dl}(l) \right] \Gamma_{CV,v}(u, l, \varepsilon) \times \quad (525)$$

$$[T_p(u) + T_n(l)] \left[\left(1 + \exp \left[\frac{\varepsilon - E_{F,p}(u)}{kT_p(u)} \right] \right)^{-1} - \left(1 + \exp \left[\frac{\varepsilon - E_{F,n}(l)}{kT_n(l)} \right] \right)^{-1} \right]$$

where $A_{CV} = \sqrt{g_C g_V} A_0$. The prefactor g_V is a fit parameter, see [Nonlocal Tunneling Parameters on page 521](#). $\Gamma_{CV,v}$ is the band-to-band tunneling probability discussed above. The modifications for zero Permeation, contacts, metals, and the BandGap option are as for [Eq. 523](#).

17: Tunneling

Nonlocal Tunneling at Interfaces, Contacts, and Junctions

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 (Eq. 525):

$$j_{CV}(l) = -q \sum_v \int_{l-\infty}^{\infty} \int_{\varepsilon}^{\infty} [R_{CV,v}(r, l, \varepsilon) - G_{CV,v}(r, l, \varepsilon)] d\varepsilon dr \quad (526)$$

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 Eq. 515 and Eq. 516) are based on the band structure near the band edges, and may not be useful in this regime. The two-band dispersion relation according Eq. 519 is a better choice. To use the two-band dispersion relation, specify the option TwoBand to eBarrierTunneling (see Table 166 on page 1137).

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 166 on page 1137).

The convective part of the heat generation in the conduction and valence bands at position u due to tunneling of carriers with energy ε to and from the v -th shifted conduction band at $l < u$ is approximated as:

$$H_{conv,CC,v}(u, l, \varepsilon) = \frac{\delta}{2} [G_{CC,v}(u)kT_n(l) - R_{CC,v}(u)kT_n(u)] \quad (527)$$

and:

$$H_{conv,CV,v}(u, l, \varepsilon) = \frac{\delta}{2} [R_{CV,v}(u)kT_p(u) - G_{CV,v}(u)kT_n(l)] \quad (528)$$

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 ε from the conduction band and the valence band at $u > l$, are $-H_{conv,CC,v}(u, l, \varepsilon)$ and $-H_{conv,CV,v}(u, l, \varepsilon)$. 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 166 on page 1137](#)), 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:

$$H_{\text{Pelt,CC},v}(u, l, \varepsilon) = [E_C(u^+) - \varepsilon][R_{\text{CC},v}(u, l, \varepsilon) - G_{\text{CC},v}(u, l, \varepsilon)] \quad (529)$$

where u^+ is infinitesimally larger than u . [Eq. 529](#) 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 [Eq. 529](#), with $E_C(u^+) - \varepsilon$ replaced by $\varepsilon - E_C(l)$.

When the Band2Band option is used (see [Table 166 on page 1137](#)), Sentaurus Device also takes into account the band-to-band terms of the Peltier part of the heat generation at point u :

$$H_{\text{Pelt,CV},v}(u, l, \varepsilon) = [E_V(u^+) - \varepsilon][R_{\text{CV},v}(u, l, \varepsilon) - G_{\text{CV},v}(u, l, \varepsilon)] \quad (530)$$

and, similarly, for the Peltier heat generation at point l .

The expressions for $H_{\text{Pelt,VV},u}$ and $H_{\text{Pelt,VC},v}$ 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.
- [3] F. Li *et al.*, “Compact Model of MOSFET Electron Tunneling Current Through Ultra-thin SiO₂ 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.

17: Tunneling

References

CHAPTER 18 Hot-Carrier Injection Models

This chapter discusses the hot-carrier injection models used in Sentaurus Device.

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)
- Fiegna's hot-carrier injection (non-Maxwellian energy distribution)
- Hot-carrier injection based on tail-energy distribution calculated from the spherical harmonic expansion of the Boltzmann transport equation
- Hot-carrier injection PMI (see [Hot-Carrier Injection on page 1022](#))

Overview

To activate the hot-carrier injection models for electrons (or holes), use the `eLucky` (`hLucky`), `eFiegna` (`hFiegna`), `TailDistribution` (`TailhDistribution`), or `PMIModel_name(electron)` (`PMIModel_name(hole)`) options to the `GateCurrent` statement in an interface-specific `Physics` section. To activate the models for both carrier types, use the `Lucky`, `Fiegna`, `TailDistribution`, or `PMIModel_name()` options. The hot-carrier injection models can be combined with all tunneling models (see [Chapter 17 on page 511](#)). Note that additional direct and Fowler–Nordheim tunneling models are not necessary in the tail distribution model as it calculates the tunneling component together with the thermionic emission term. 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 512](#)).

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

18: Hot-Carrier Injection Models

Overview

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 53](#)).

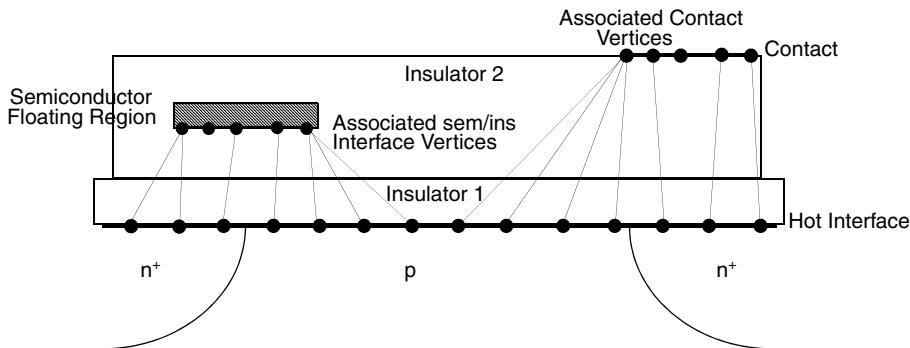


Figure 53 Mapping of hot interface vertices to associated contact or semiconductor–insulator interface vertices

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 as defined by [Eq. 117, p. 213](#). 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 545](#)). 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 in equilibrium for carrier concentrations.

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, and $N_{C,V}^{WB}/N_{C,V}^{FG}$ 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"}
    ...
}

Physics (MaterialInterface="OxideAsSemiconductor/PolySi") {
    Heterointerface                      # required by GeneralizedFG
}

```

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–SiO₂ barrier (E_B).

The height is a function of the insulator field F_{ins} and, at any point along the interface, it can be written as:

$$E_B = \begin{cases} E_{B0} - \alpha q |F_{\text{ins}}|^{\frac{1}{2}} - \beta q |F_{\text{ins}}|^{\frac{2}{3}} & F_{\text{ins}} < 0 \\ E_{B0} - \alpha q |F_{\text{ins}}|^{\frac{1}{2}} - \beta q |F_{\text{ins}}|^{\frac{2}{3}} + V_{\text{ins}} & F_{\text{ins}} > 0 \end{cases} \quad (531)$$

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–SiO₂ 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 β , so it can be considered a fitting parameter.

In addition, all hot-carrier models contain a probability P_{ins} of scattering in the image-force potential well:

$$P_{\text{ins}} = \begin{cases} \exp\left(-\frac{x_0}{\lambda_{\text{ins}}}\right) & F_{\text{ins}} < 0 \\ \exp\left(-\frac{t_{\text{ins}} - x_0}{\lambda_{\text{ins}}}\right) & F_{\text{ins}} > 0 \end{cases} \quad (532)$$

where λ_{ins} is the scattering mean free path in the insulator, t_{ins} is the distance between the vertex at the hot interface and the closest associated vertex, and the distance x_0 is given as:

$$x_0 = \min\left(\sqrt{\frac{q}{16\pi\varepsilon_{\text{ins}}|F_{\text{ins}}|}}, \frac{t_{\text{ins}}}{2}\right) \quad (533)$$

In the above expression, ε_{ins} is the dielectric constant of the insulator.

Effective Field

The lucky electron model and Fiega model have an effective electric field F_{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 331](#).
- With recomputation of the carrier temperature of the hydrodynamic simulation (switched on by the keyword `CarrierTempDrive`). See [Avalanche Generation with Hydrodynamic Transport on page 331](#).
- 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`, `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]:

$$I_g = \iint J_n(x, y) P_s P_{\text{ins}} \left(\int_{E_B}^{\infty} P_{\varepsilon} P_r d\varepsilon \right) dx dy \quad (534)$$

where P_s is the probability that the electron will travel a distance y to the interface without losing any energy, $P_{\varepsilon} d\varepsilon$ is the probability that the electron has energy between ε and $\varepsilon + d\varepsilon$, P_{ins} is the probability of scattering in the image force potential well ([Eq. 532](#)), and P_r is the probability that the electron will be redirected. These probabilities are given by the expressions:

$$P_r(\varepsilon) = \frac{1}{2\lambda_r} \left(1 - \sqrt{\frac{E_B}{\varepsilon}} \right) \quad (535)$$

$$P_s(y) = \exp\left(-\frac{y}{\lambda}\right) \quad (536)$$

18: Hot-Carrier Injection Models

Fiegna Hot-Carrier Injection

$$P_\varepsilon(\varepsilon) = \frac{1}{\lambda F_{\text{eff}}} \exp\left(-\frac{\varepsilon}{\lambda F_{\text{eff}}}\right) \quad (537)$$

where λ is the scattering mean free path in the semiconductor, λ_r is redirection mean free path, F_{eff} is the effective electric field, see [Effective Field on page 535](#). E_{B0} is the height of the semiconductor–insulator barrier. The model coefficients and their defaults are given in [Table 88](#).

They can be changed in the parameter file in the section:

```
LuckyModel { ... }
```

Table 88 Default coefficients for lucky electron model

Symbol	Parameter name (Electrons)	Default value (Electrons)	Parameter name (Holes)	Default value (Holes)	Unit
λ	eLsem	8.9×10^{-7}	hLsem	1.0×10^{-7}	cm
λ_{ins}	eLins	3.2×10^{-7}	hLins	3.2×10^{-7}	cm
λ_r	eLsemR	6.2×10^{-6}	hLsemR	6.2×10^{-6}	cm
E_{B0}	eBar0	3.1	hBar0	4.7	eV
α	eBL12	2.6×10^{-4}	hBL12	2.6×10^{-4}	$(V \cdot \text{cm})^{1/2}$
β	eBL23	3.0×10^{-5}	hBL23	3.0×10^{-5}	$(V \cdot \text{cm}^2)^{1/3}$

Fiegna Hot-Carrier Injection

The total hot-carrier injection current according to the Fiegna model [2] can be written as:

$$I_g = q \int P_{\text{ins}} \left(\int_{E_{B0}}^{\infty} v_{\perp}(\varepsilon) f(\varepsilon) g(\varepsilon) d\varepsilon \right) ds \quad (538)$$

where ε is the electron energy, E_{B0} is the height of the semiconductor–insulator barrier, v_{\perp} is the velocity normal to the interface, $f(\varepsilon)$ is the electron energy distribution, $g(\varepsilon)$ is the density-of-states of the electrons, P_{ins} is the probability of scattering in the image force potential well as described by [Eq. 532](#), and $\int ds$ 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:

$$f(\varepsilon) = A \exp\left(-\chi \frac{\varepsilon^3}{F_{\text{eff}}^{1.5}}\right) \quad (539)$$

Therefore, the gate current can be rewritten as:

$$I_g = q \frac{A}{3\chi} \int P_{\text{ins}} n \frac{F_{\text{eff}}^{3/2}}{\sqrt{E_B}} e^{-\frac{\chi E_B^3}{F_{\text{eff}}^{3/2}}} ds \quad (540)$$

where F_{eff} is an effective field (see [Effective Field on page 535](#)).

The coefficients and their defaults are given in [Table 89](#).

Table 89 Coefficients for Fiegna model

	A [cm/s/eV ^{2.5}]	χ [(V/cm · eV) ^{1.5}]	λ_{ins} [cm]	E_{B0} [eV]	α [(V · cm) ^{1/2}]	β [(V · cm ²) ^{1/3}]
Electrons	4.87×10^2	1.3×10^8	3.2×10^{-7}	3.1	2.6×10^{-4}	1.5×10^{-5}
Holes	4.87×10^2	1.3×10^8	3.2×10^{-7}	4.7	2.6×10^{-4}	1.5×10^{-5}

The above coefficients can be changed in the parameter file in the `FiegnaModel` section. Coefficients A , χ , λ_{ins} , E_{B0} , α , and β correspond to `eA`, `eChi`, `eLins`, `eBar0`, `eBL12`, and `eBL23` for electrons and `hA`, `hChi`, `hLins`, `hBar0`, `hBL12`, and `hBL23` for holes in the parameter file.

Tail Distribution Hot-Carrier Injection

To obtain the hot-carrier injection current, accurate knowledge of the nonequilibrium electron-energy distribution is required. The tail distribution hot-carrier injection model calculates the hot-carrier injection current using the nonequilibrium energy distribution f obtained from the lowest-order spherical harmonic expansion (SHE) of the semiclassical Boltzmann transport equation (BTE).

Tail Distribution Model

The lowest-order SHE of the BTE can be written as [3]:

$$-\nabla \cdot \left[\frac{v^2(\vec{r}, \varepsilon_t)}{3} \tau(\vec{r}, \varepsilon_t) g(\vec{r}, \varepsilon_t) \nabla f(\vec{r}, \varepsilon_t) \right] = g(\vec{r}, \varepsilon_t) s(\vec{r}, \varepsilon_t) \quad (541)$$

where:

- ε_t is the total energy including the conduction band energy E_C and the kinetic energy ε .
- v is the magnitude of the electron velocity.
- $1/\tau$ is the total scattering rate.
- g is the density-of-states.
- s is the net in-scattering rate due to inelastic scattering and generation–recombination processes.

The analytic nonparabolic band-structure model gives [4]:

$$\frac{v^2(\varepsilon)}{3} = \frac{2\varepsilon(1+\alpha\varepsilon)}{3m_c(1+2\alpha\varepsilon)^2} \quad (542)$$

$$g(\varepsilon) = \frac{2\pi(2m_n)^{3/2}}{h^3} [\varepsilon(1+\alpha\varepsilon)]^{1/2} (1+2\alpha\varepsilon) \quad (543)$$

where $\varepsilon = \varepsilon_t - E_C(\vec{r})$ is the kinetic energy, α is the nonparabolicity factor, m_c is the conductivity effective mass, m_n is the density-of-states effective mass, and h is Planck's constant.

The total scattering rate and the net in-scattering rate can be written as:

$$\frac{1}{\tau(\varepsilon)} = \frac{1}{\tau_c(\varepsilon)} + \frac{1}{\tau_{ii}(\varepsilon)} + \frac{1}{\tau_{ac}(\varepsilon)} + \frac{1}{\tau_{ope}(\varepsilon)} + \frac{1}{\tau_{opa}(\varepsilon)} \quad (544)$$

$$s(\varepsilon) = \frac{f_{loc}(\varepsilon) - f(\varepsilon)}{\tau_{ii}(\varepsilon)} + \frac{f(\varepsilon - \varepsilon_{op}) \exp\left(-\frac{\varepsilon_{op}}{kT}\right) - f(\varepsilon)}{\tau_{ope}(\varepsilon)} + \frac{f(\varepsilon + \varepsilon_{op}) \exp\left(\frac{\varepsilon_{op}}{kT}\right) - f(\varepsilon)}{\tau_{opa}(\varepsilon)} - \frac{R_{net}f_{loc}(\varepsilon)}{n} \quad (545)$$

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.

- $1/\tau_{\text{ope}}$ is the scattering rate due to optical phonon emissions.
- $1/\tau_{\text{opa}}$ is the scattering rate due to optical phonon absorptions.
- ε_{op} is the optical phonon energy.
- R_{net} is the net recombination rate.
- n is the electron density.
- $f_{\text{loc}} = \exp[(E_{F,n} - E_C - \varepsilon)/kT]$ is the local equilibrium distribution function.

The Coulomb scattering rate can be written as [5]:

$$\frac{1}{\tau_c(\varepsilon)} = \frac{q^4(1+2\alpha\varepsilon)(N_D + N_A)\zeta(N_D, N_A)}{16\pi\varepsilon_{\text{sem}}^2\varepsilon(1+\alpha\varepsilon)\sqrt{2m_n\varepsilon(1+\alpha\varepsilon)}} \left[\ln(1+b) - \frac{b}{1+b} \right] \quad (546)$$

$$b(\varepsilon) = \frac{32\pi^2 m_n \varepsilon (1+\alpha\varepsilon) k T \varepsilon_{\text{sem}}}{q^2 h^2 (n+p)} \quad (547)$$

$$\zeta(N_D, N_A) = \begin{cases} \zeta_{\text{major}}(N_D) & N_D > N_A \\ \zeta_{\text{minor}}(N_A) & N_D < N_A \end{cases} \quad (548)$$

where ε_{sem} is the dielectric constant of a semiconductor material, and ζ_{major} and ζ_{minor} are tabulated fitting functions introduced to match the experimental low-field mobility curve as a function of majority and minority doping concentrations, respectively.

The impact ionization scattering rate can be written as [5]:

$$\frac{1}{\tau_{ii}(\varepsilon)} = \begin{cases} \left(\frac{\varepsilon - \varepsilon_{ii,1}}{1 \text{ eV}}\right)^3 s_{ii,1} & \varepsilon_{ii,1} < \varepsilon < \varepsilon_{ii,3} \\ \left(\frac{\varepsilon - \varepsilon_{ii,2}}{1 \text{ eV}}\right)^2 s_{ii,2} & \varepsilon > \varepsilon_{ii,3} \end{cases} \quad (549)$$

where $s_{ii,1}$ and $s_{ii,2}$ are the impact ionization coefficients, and $\varepsilon_{ii,1}$, $\varepsilon_{ii,2}$, and $\varepsilon_{ii,3}$ are the reference energies.

The acoustic-phonon and optical-phonon scattering rates can be written as [4][5]:

$$\frac{1}{\tau_{ac}(\varepsilon)} = \frac{4\pi^2 k T D_{ac}^2}{h \rho c_L^2} g(\varepsilon) \quad (550)$$

18: Hot-Carrier Injection Models

Tail Distribution Hot-Carrier Injection

$$\frac{1}{\tau_{\text{ope}}(\varepsilon)} = \frac{hD_{\text{op}}^2}{2\rho\varepsilon_{\text{op}}} (N_{\text{op}} + 1)g(\varepsilon - \varepsilon_{\text{op}}) \quad (551)$$

$$\frac{1}{\tau_{\text{opa}}(\varepsilon)} = \frac{hD_{\text{op}}^2}{2\rho\varepsilon_{\text{op}}} N_{\text{op}} g(\varepsilon + \varepsilon_{\text{op}}) \quad (552)$$

where:

- D_{ac} and D_{op} are the deformation potentials for acoustic and g-type optical phonons, respectively.
- ρ is the mass density.
- c_L is the sound velocity.
- $N_{\text{op}} = [\exp(\varepsilon_{\text{op}}/kT) - 1]^{-1}$ is the phonon number.

Injection Model

When the energy distribution is available, the total hot-carrier injection current is obtained from:

$$\begin{aligned} I_g &= \frac{2qg_v}{h^3} \int P_{\text{ins}} \left[\int \Gamma(\varepsilon_{\perp}) v_{\perp}(\varepsilon_{\perp}, \varepsilon) f(\varepsilon) d\vec{p} \right] ds \\ &= \frac{4\pi q g_v m_n^{3/2}}{h^3 m_c^{1/2}} \int P_{\text{ins}} \left[\int_0^\infty \int_0^\infty \frac{\Gamma(\varepsilon_{\perp}) f(\varepsilon)}{1 + 2\alpha\varepsilon} d\gamma d\delta \right] ds \end{aligned} \quad (553)$$

where:

- g_v is the valley degeneracy factor.
- \vec{p} is the crystal momentum.
- P_{ins} is the probability of electrons moving from the interface to the barrier peak without scattering.
- Γ is the transmission coefficient obtained from the WKB approximation including the image-potential barrier lowering.
- ε_{\perp} is the perpendicular kinetic energy.
- v_{\perp} is the perpendicular velocity.
- $\int ds$ is an integral along the semiconductor–insulator interface.

In terms of the integration variables γ and δ , ε and ε_{\perp} can be written as:

$$\varepsilon = \frac{-1 + \sqrt{1 + 4\alpha(\gamma + \delta)}}{2\alpha} \quad (554)$$

$$\varepsilon_{\perp} = \frac{-1 + \sqrt{1 + 4\alpha\gamma}}{2\alpha} \quad (555)$$

The transmission coefficient can be written as:

$$\Gamma(\varepsilon_{\perp}) = \exp\left(-\frac{2}{\hbar} \int_0^{t_{\text{ins}}} \sqrt{2m_{\text{ins}}[E_B(r) - \varepsilon_{\perp}]} \Theta[E_B(r) - \varepsilon_{\perp}] dr\right) \quad (556)$$

$$E_B(r) = E_{B0} + qF_{\text{ins}}r + E_{\text{im}}(r) \quad (557)$$

$$E_{\text{im}}(r) = -\frac{q^2}{16\pi\varepsilon_{\text{ins}}} \sum_{n=0}^{\infty} \left(\frac{\varepsilon_{\text{sem}} - \varepsilon_{\text{ins}}}{\varepsilon_{\text{sem}} + \varepsilon_{\text{ins}}}\right)^{2n+1} \left[\frac{1}{nt_{\text{ins}} + r} + \frac{1}{(n+1)t_{\text{ins}} - r} - \frac{2}{(n+1)t_{\text{ins}}} \left(\frac{\varepsilon_{\text{sem}} - \varepsilon_{\text{ins}}}{\varepsilon_{\text{sem}} + \varepsilon_{\text{ins}}}\right) \right] \quad (558)$$

where m_{ins} is the insulator effective mass, and E_{B0} is the barrier height.

Finally, the distance from the interface to the barrier peak r_0 is calculated numerically from Eq. 557, and P_{ins} is directly obtained from:

$$P_{\text{ins}} = \exp\left(-\frac{r_0}{\lambda_{\text{ins}}}\right) \quad (559)$$

instead of using Eq. 532, p. 534.

Using Tail Distribution Hot-Carrier Injection Model

The electron-energy distribution function is calculated from Eq. 541, p. 538 in the semiconductor regions specified by the global, region-specific, or material-specific Physics section:

```
Physics{
    TailDistribution
}
```

Similarly, for the hole-energy distribution function, specify TailhDistribution in the Physics section.

[Eq. 541, p. 538](#) is a coupled energy-dependent conservation equation with diffusion and source terms, which is currently solved iteratively by using the blockwise successive over-relaxation (SOR) method. The linear solver, the number of iterations, and the SOR parameter can be accessed by the keywords TailDistributionMethod, TailDistributionIterations, and TailDistributionSORParameter in the global Math section. The default values are:

```
Math {
    TailDistributionMethod=super
    TailDistributionIterations=20
    TailDistributionSORParameter=1.1
}
```

The parameters for the tail distribution model are available in the TailDistribution parameter set. [Table 90](#) lists the coefficients of models and their default values.

NOTE The optical phonon energy ε_{op} is currently used as an energy grid spacing. Therefore, the same optical phonon energy must be used in the simulation domain.

Table 90 Default parameters for tail distribution model

Symbol	Parameter name	Electrons	Holes	Unit
ρ	rho	2.329		g/cm^3
ε_{sem}	epsilon	11.7		ε_0
ε_{ins}	eps_ins	3.9		ε_0
m_c	m_s	0.26	0.26	m_0
m_n	m_dos	0.328	0.328	m_0
m_{ins}	m_ins	0.5	0.77	m_0
α	alpha	0.5	0.5	eV^{-1}
g_v	g	6	6	1
E_{B0}	E_barrier	3.15	4.73	eV
λ_{ins}	Lins	3.2×10^{-7}	3.2×10^{-7}	cm
D_{ac}/c_L	Dac_cl	9.8808×10^{-6}	9.8808×10^{-6}	eVs/cm
D_{op}	Dop	1.1×10^9	1.1×10^9	eV/cm
ε_{op}	HbarOmega	0.06	0.06	eV
$s_{ii,1}$	ii_rate1	1.49×10^{11}	1.49×10^{11}	1/s
$s_{ii,2}$	ii_rate2	1.13×10^{12}	1.13×10^{12}	1/s
$\varepsilon_{ii,1}$	ii_energy1	1.128	1.128	eV

Table 90 Default parameters for tail distribution model

Symbol	Parameter name	Electrons	Holes	Unit
$\varepsilon_{ii,2}$	ii_energy2	1.572	1.572	eV
$\varepsilon_{ii,3}$	ii_energy3	1.75	1.75	eV

Table 91 lists the coefficients and the default values of the tabulated doping-dependent fitting parameters ζ_{major} and ζ_{minor} for electrons. The default values for holes (`hfit`) are the same.

Table 91 Default parameters for doping-dependent functions ζ_{major} and ζ_{minor}

Doping	Parameter name	ζ_{major}	ζ_{minor}	Unit
$10^{15.00}/\text{cm}^3$	<code>efit(0)</code>	1.00	2.30	1
$10^{15.25}/\text{cm}^3$	<code>efit(1)</code>	1.10	2.30	1
$10^{15.50}/\text{cm}^3$	<code>efit(2)</code>	1.25	2.30	1
$10^{15.75}/\text{cm}^3$	<code>efit(3)</code>	1.40	2.30	1
$10^{16.00}/\text{cm}^3$	<code>efit(4)</code>	1.45	2.30	1
$10^{16.25}/\text{cm}^3$	<code>efit(5)</code>	1.50	2.35	1
$10^{16.50}/\text{cm}^3$	<code>efit(6)</code>	1.65	2.40	1
$10^{16.75}/\text{cm}^3$	<code>efit(7)</code>	1.85	2.45	1
$10^{17.00}/\text{cm}^3$	<code>efit(8)</code>	2.00	2.50	1
$10^{17.25}/\text{cm}^3$	<code>efit(9)</code>	2.40	2.40	1
$10^{17.50}/\text{cm}^3$	<code>efit(10)</code>	2.50	2.30	1
$10^{17.75}/\text{cm}^3$	<code>efit(11)</code>	2.80	2.20	1
$10^{18.00}/\text{cm}^3$	<code>efit(12)</code>	3.30	2.10	1
$10^{18.25}/\text{cm}^3$	<code>efit(13)</code>	3.40	1.85	1
$10^{18.50}/\text{cm}^3$	<code>efit(14)</code>	3.80	1.75	1
$10^{18.75}/\text{cm}^3$	<code>efit(15)</code>	3.80	1.60	1
$10^{19.00}/\text{cm}^3$	<code>efit(16)</code>	4.00	1.61	1
$10^{19.25}/\text{cm}^3$	<code>efit(17)</code>	4.80	1.70	1
$10^{19.50}/\text{cm}^3$	<code>efit(18)</code>	6.90	1.95	1
$10^{19.75}/\text{cm}^3$	<code>efit(19)</code>	9.00	2.80	1
$10^{20.00}/\text{cm}^3$	<code>efit(20)</code>	10.7	3.90	1

Visualizing Tail Distribution

To plot the position-dependent electron-energy distribution function f for each energy grid, specify `TaileDistribution/SpecialVector` in the `Plot` section:

```
Plot{
  ...
  TaileDistribution/SpecialVector
}
```

Similarly, for the hole-energy distribution function, specify `TailhDistribution/SpecialVector`. The column i of the special vector represents the distribution function at $\epsilon = (i + 1)\epsilon_{\text{op}}$. For example, `TaileDistribution_C2` represents f at $\epsilon = 3\epsilon_{\text{op}}$.

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 `TaileDistribution`:

```
File {
  ...
  TaileDistribution = "edist"
}
```

Plotting is activated by including the `TaileDistributionPlot` section (similar to the `CurrentPlot` section) in the command file with a set of coordinates of positions:

```
TaileDistributionPlot {
  (-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 `TailhDistribution` in the `File` section and include the `TailhDistributionPlot` section in the command file.

Carrier Injection with Explicitly Evaluated Boundary Conditions for Continuity Equations

Hot-carrier current can be added as an interface boundary condition for continuity equations in adjacent semiconductor regions. A typical structure (see [Figure 54](#)) 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 531](#). This feature is available only for transient simulations and is especially useful for writing and erasing memory cells.

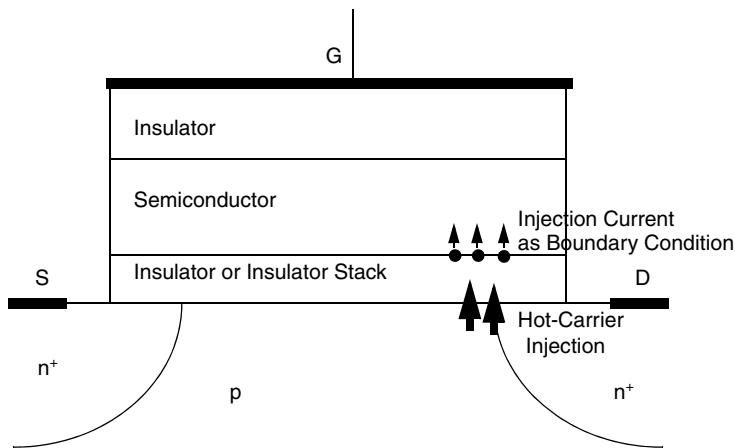


Figure 54 Injection of hot-carrier current in a MOSFET structure

At each time step after the solution is computed, the hot-carrier injection (HCI) currents are post-evaluated using the solution. For the next time step, the HCI 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.

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.

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.

18: Hot-Carrier Injection Models

References

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 531](#). 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_sem12/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. Fiegnan *et al.*, “Simple and Efficient Modeling of EPROM Writing,” *IEEE Transactions on Electron Devices*, vol. 38, no. 3, pp. 603–610, 1991.
- [3] 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.
- [4] 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.
- [5] C. Jungemann and B. Meinerzhagen, *Hierarchical Device Simulation: The Monte-Carlo Perspective*, Vienna: Springer, 2003.

This chapter presents information about the simulation of devices containing arbitrary materials and heterojunctions in Sentaurus Device.

In addition, it is possible to use different physical models and different parameter sets in different regions and materials of the device. This is available for both homogenous and heterojunction devices as described in [Region-specific and Material-specific Physics on page 55](#). To use the default material parameters and the same set of models for all regions and materials, special specifications are not required for the input file. The program automatically uses appropriate parameters for all materials defined in the geometry file.

Physics Models and Differential Equations

Most of the models described in this manual can be applied to both homogenous semiconductors and heterostructures. Conversely, some models are applicable only for heteromaterials. In this section, the most important models are described:

- Transport equations and energy balance equations contain terms such as $\nabla\chi$, ∇E_g , and $\nabla \ln(m_n)$, which are equal to 0 for homogeneous structures (the term ∇E_g can be nonzero if the bandgap narrowing effect is taken into account), but are nonzero and crucial if a heterostructure is simulated.
- Thermionic emission models at abrupt heterointerfaces (see [Thermionic Emission Current on page 559](#)).
- Tunneling at heterointerfaces (see [Chapter 17 on page 511](#)).

Mole-Fraction Materials

Sentaurus Device reads the file `Molefraction.txt` 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. Sentaurus Device checks if the environment variable `SDEVICEDEB` is defined. This variable should contain a directory or a list of directories separated by spaces or colons, for example:

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

Sentaurus Device scans the directories in the given order until `Molefraction.txt` is found.

NOTE The environment variable `SDEVICEDEB` is also used to locate libraries of material parameters (see [Library of Materials on page 117](#)).

3. If the environment variables `STROOT` and `STRELEASE` are defined, Sentaurus Device tries to read the file:

```
$STROOT/tcad/$STRELEASE/lib/sdevice/MaterialDB/Molefraction.txt
```

4. If these previous strategies are unsuccessful, Sentaurus Device uses the built-in defaults that follow.

The default `Molefraction.txt` file has the following content:

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

```

In order 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

```

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

```

AlGaInAs (x=0, y=0) = InAs
AlGaInAs (x=1, y=0) = AlAs
AlGaInAs (x=0, y=1) = GaAs

```

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

NOTE All side and corner materials must appear in `datexcodes.txt`, and their mole dependencies must be specified in `Molefraction.txt` (see [Model Hierarchy on page 121](#)).

19: Heterostructure Device Simulation

Mole-Fraction Specification

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

Mole-Fraction Specification

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

- In the data file (`<name>.tdr`) of the device structure
- Internally, in the `Physics` section of the `des.cmd` input 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 input file. 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 214 on page 1171](#).

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)
}
```

Composition-dependent Models

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

- Epsilon (also available for compound insulators)
- LatticeHeatCapacity (also available for compound insulators)
- Kappa (lattice thermal conductivity, also available for compound insulators)
- EnergyRelaxationTime
- Bandgap
- eDOSMass
- hDOSMass
- ConstantMobility
- DopingDependence (in the mobility models)
- HighFieldDependence (in the mobility models)
- Enormal (in the mobility models)
- ToCurrentEnormal (in the mobility models)
- PhuMob (in the mobility models)
- vanOverstraetenMan (impact ionization model)
- SchroedingerParameters
- AbsorptionCoefficient
- QWStrain (see [Syntax for Quantum-Well Strain on page 821](#))
- RefractiveIndex (also available for compound insulators)
- Radiative recombination
- Shockley–Read–Hall recombination
- Auger recombination
- Piezoelectric polarization

19: Heterostructure Device Simulation

Ternary Semiconductor Composition

- QuantumPotentialParameters
- Deformation potential (elasticity modulus, the parameters for the electron band: `xis, dbs, xiu, xid` and, for hole band: `adp, bdp, ddp, dso`)

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. If this is required, specify the (fixed) values for the parameter (for example, `Eg0=1.53`) and delete 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:

$$\begin{aligned} P &= P_{i-1} + A \cdot \Delta x + B_i \cdot \Delta x^2 + C_i \cdot \Delta x^3 \\ A &= \frac{\Delta P_i}{\Delta x_i} - B_i \cdot \Delta x_i - C_i \cdot \Delta x_i^2 \\ \Delta P_i &= P_i - P_{i-1} \\ \Delta x_i &= x_i - x_{i-1} \\ \Delta x &= x - x_{i-1} \end{aligned} \tag{560}$$

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 $x = 1$):

$$P = (1 - x)P_{x0} + xP_{x1} \quad (561)$$

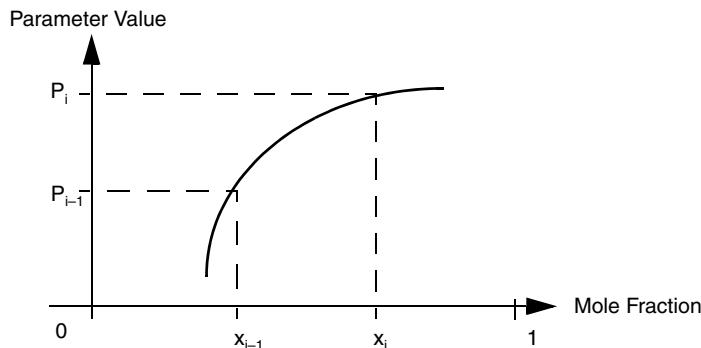


Figure 55 Parameter value as a function of mole fraction

Example 1: Specifying Electron Effective Mass

This example shows the specification of the parameters that define the electron effective mass, in the default section of the parameter file, for the material GaAs:

```
eDOSMass
{
    * For effective mass specification Formula1 (me approximation):
    * or Formula2 (Nc300) can be used:
    Formula= 1      # [1]
    * Formula1:
    * me/m0 = [ (6 * mt)^2 * m1 ]^(1/3) + mm
    * mt = a[Eg(0)/Eg(T)]
    * Nc(T) = 2(2pi*kB/h_Plank^2*me*T)^3/2 = 2.540e19 ((me/m0)*T)^3/2
    a = 0.1905      # [1]
    m1 = 0.9163     # [1]
    mm = 0.0000e+00 # [1]
    * Formula2:
    * me/m0 = (Nc300/2.540e19)^2/3
    * Nc(T) = Nc300 * (T/300)^3/2
    Nc300 = 2.8000e+19 # [cm^-3]
}
```

You can select between `Formula1` and `Formula2`. By default, one value (`Formula=...`) is chosen for each material. If necessary, the formula is also changeable in the parameter file.

For mole fraction-dependent materials, certain model parameters are specified as a function of the mole fraction. The mole fraction dependence is implemented as a piecewise polynomial approximation, up to the third order.

Example 2: 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 $x = 1$.

NOTE Although this example uses a linear interpolation over the whole interval $[0, 1]$, a higher order polynomial can be selected by specifying the appropriate parameters. Additional mole fraction intervals can also be introduced as shown in the next example.

Example 3: 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 previous example, two intervals were 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 polynomial:

$$F + A(X - \text{Xmin}(I)) + B(X - \text{Xmin}(I))^2 + C(X - \text{Xmin}(I))^3 \quad (562)$$

are determined from the values at both ends of the intervals, while 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 Band Gap 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]
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:

$$AB_xC_yD_z \quad (563)$$

where A is a group III element, and B , C , and D are group V elements (usually listed according to increasing atomic number). Conversely, a 3:1 III–V quaternary alloy can be described as:

$$A_xB_yC_zD \quad (564)$$

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:

$$x + y + z = 1 \quad (565)$$

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 $P(A_xB_yC_zD)$ of a 3:1 III–V quaternary alloy as a weighted sum of the corresponding ternary values:

$$P(A_xB_yC_zD) = \frac{xyP(A_{1-u}B_uD) + yzP(B_{1-v}C_vD) + xzP(A_{1-w}C_wD)}{xy + yz + xz} \quad (566)$$

where:

$$u = \frac{1-x+y}{2}, v = \frac{1-y+z}{2}, w = \frac{1-x+z}{2} \quad (567)$$

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:

$$A_xB_{1-x}C_yD_{1-y} \quad (568)$$

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, we mention the material $\text{In}_{1-x}\text{Ga}_x\text{As}_y\text{P}_{1-y}$.

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

$$P(A_xB_{1-x}C_yD_{1-y}) = \frac{x(1-x)(yP(A_xB_{1-x}C) + (1-y)P(A_xB_{1-x}D)) + y(1-y)(xP(AC_yD_{1-y}) + (1-x)P(BC_yD_{1-y}))}{x(1-x) + y(1-y)} \quad (569)$$

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 `Material`. Assume that no default parameters are defined for this material and a given physical model `Model`.

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.

Sometimes, it is difficult to analyze such a parameter file to obtain a real value of physical models (for example, the band gap) for certain 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.

Abrupt and Graded Heterojunctions

Sentaurus Device is designed to support both abrupt and graded heterojunctions, with an arbitrary mole fraction distribution. As previously described, a piecewise polynomial interpolation is used for interpolation of parameters of the physical models over the mole fraction x . 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](#)) 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.

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 56](#) shows a distribution of 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 (δE_C).

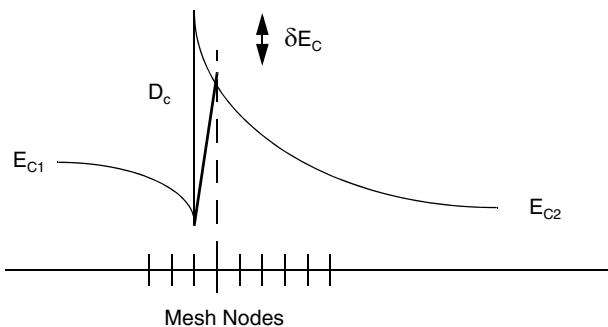


Figure 56 Heterostructure barrier representation with and without double points

Thermionic Emission Current

Conventional transport equations cease to be valid at a heterojunction interface, and currents and energy fluxes at the abrupt interface between two materials are better defined by the interface condition at the heterojunction. In defining thermionic current and thermionic energy flux, Sentaurus Device follows the literature [7].

Using Thermionic Emission Current

To activate the thermionic current model for electrons at a region-interface (material-interface) heterojunction, the keyword `eThermionic` must be specified in the appropriate region-interface (material-interface) `Physics` section, for example:

```
Physics (MaterialInterface="GaAs/AlGaAs") {  
    eThermionic  
}
```

Similarly, to activate thermionic current for holes, the keyword `hThermionic` must be specified. The keyword `Thermionic` activates the thermionic emission model for both electrons and holes. If any of these keywords is specified in the `Physics` section for a region `Region . 0`, where `Region . 0` is a semiconductor, the appropriate model will be applied to each `Region . 0`-semiconductor interface.

For small particle and energy fluxes across the interface, the condition of continuous quasi-Fermi level and carrier temperature is sometimes used. This option is activated by the keyword `Heterointerface` in the appropriate `Physics` section. In realistic transistors, such an approach may lead to unsatisfactory results [8].

You can change the coefficients of the thermionic emission model in the `ThermionicEmission` parameter set in an interface-specific section in the parameter file:

```
RegionInterface = "regionA/regionB" {  
    ThermionicEmission {  
        A = 2, 2 # [1]  
        B = 4, 4 # [1]  
        C = 1, 1 # [1]  
    }  
}
```

Thermionic Emission Model

Assume that at the heterointerface between materials 1 and 2, the conduction edge jump is positive, that is $\Delta E_C > 0$, where $\Delta E_C = E_{C,2} - E_{C,1}$ (that is, $\chi_1 > \chi_2$). If $J_{n,2}$ and $S_{n,2}$ are the electron current density and electron energy flux density entering material 2, and $J_{n,1}$ and $S_{n,1}$ are the electron current density and electron energy flux density leaving material 1, the interface condition can be written as:

$$J_{n,2} = J_{n,1} \quad (570)$$

$$J_{n,2} = a_n q \left[v_{n,2} n_2 - \frac{m_{n,2}}{m_{n,1}} v_{n,1} n_1 \exp\left(-\frac{\Delta E_C}{kT_{n,1}}\right) \right] \quad (571)$$

$$S_{n,2} = S_{n,1} + \frac{c_n}{q} J_{n,2} \Delta E_C \quad (572)$$

$$S_{n,2} = -b_n \left[v_{n,2} n_2 k T_{n,2} - \frac{m_{n,2}}{m_{n,1}} v_{n,1} n_1 k T_{n,1} \exp\left(-\frac{\Delta E_C}{kT_{n,1}}\right) \right] \quad (573)$$

where the ‘emission velocities’ are defined as:

$$v_{n,i} = \sqrt{\frac{kT_{n,i}}{2\pi m_{n,i}}} \quad (574)$$

and by default, the coefficients in the above equations are $a_n = 2$, $b_n = 4$, and $c_n = 1$, which corresponds to the literature [7]. Similar equations for the hole thermionic current and hole thermionic energy flux are presented below:

$$J_{p,2} = J_{p,1} \quad (575)$$

$$J_{p,2} = -a_p q \left[v_{p,2} p_2 - \frac{m_{p,2}}{m_{p,1}} v_{p,1} p_1 \exp\left(\frac{\Delta E_V}{kT_{p,1}}\right) \right] \quad (576)$$

$$S_{p,2} = S_{p,1} + \frac{c_p}{q} J_{p,2} \Delta E_V \quad (577)$$

$$S_{p,2} = -b_p \left[v_{p,2} p_2 k T_{p,2} - \frac{m_{p,2}}{m_{p,1}} v_{p,1} p_1 k T_{p,1} \exp\left(\frac{\Delta E_V}{kT_{p,1}}\right) \right] \quad (578)$$

$$v_{p,i} = \sqrt{\frac{kT_{p,i}}{2\pi m_{p,i}}} \quad (579)$$

An equivalent set of equations are used if Fermi carrier statistics is selected.

Gaussian Transport Across Organic Heterointerfaces

A thermionic-like current boundary condition has been introduced to correctly account for carrier transport across organic heterointerfaces. An organic heterointerface is defined in this context as an heterointerface with the Gaussian density-of-states (DOS) model (see [Gaussian Density-of-States for Organic Semiconductors on page 240](#)) activated in both regions that are adjacent to the heterointerface.

Using Gaussian Transport at Organic Heterointerfaces

The model is activated by switching to the Gaussian DOS model in both regions of the heterointerface that are meant to be organic:

```
Physics(Region="OrganicRegion_1") {
    EffectiveMass(GaussianDOS)
}

Physics(Region="OrganicRegion_2") {
    EffectiveMass(GaussianDOS)
}
```

and then specifying the keyword `Organic_Gaussian` as an option for the Thermionic model in the `Physics` section of the organic heterointerface:

```
Physics(RegionInterface="OrganicRegion_1/OrganicRegion_2") {
    Thermionic(Organic_Gaussian)
}
```

This syntax also automatically switches on the double points at the organic heterointerface.

The parameters $v_{n,\text{org}}$ and $v_{p,\text{org}}$ in [Eq. 581, p. 562](#) and [Eq. 583, p. 562](#) can be adjusted in the `ThermionicEmission` section of the parameter file (their default values are $1 \times 10^6 \text{ cm/s}$):

```
RegionInterface="OrganicRegion_1/OrganicRegion_2" {
    ThermionicEmission {
        vel_org = 1e7, 1e7    # [cm/s]
    }
}
```

Gaussian Transport at Organic Heterointerface Model

Assuming a positive conduction edge jump and a negative valence edge jump from material 1 to material 2, the boundary conditions at the organic heterointerface are given by:

$$J_{n,2} = J_{n,1} \quad (580)$$

$$J_{n,2} = v_{n,\text{org}} q \left(n_2 - n_1 \exp\left(-\frac{\Delta E_C}{kT_{n,1}}\right) \right) \quad (581)$$

$$J_{p,2} = J_{p,1} \quad (582)$$

$$J_{p,2} = v_{p,\text{org}} q \left(p_2 - p_1 \exp\left(-\frac{\Delta E_V}{kT_{p,1}}\right) \right) \quad (583)$$

where:

$J_{n,2}$ and $J_{n,1}$ are the electron current densities entering material 2 and leaving material 1, respectively.

$J_{p,2}$ and $J_{p,1}$ are the hole current densities leaving material 2 and entering material 1, respectively.

$\Delta E_C = d_{C,2} - d_{C,1}$ where $d_{C,2}$ and $d_{C,1}$ are the distances of the Gaussian distribution peaks to the conduction band edges.

$\Delta E_V = d_{V,2} - d_{V,1}$ where $d_{V,2}$ and $d_{V,1}$ are the distances of the Gaussian distribution peaks to the valence band edges.

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.

- [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.
- [7] D. Schroeder, *Modelling of Interface Carrier Transport for Device Simulation*, Wien: Springer, 1994.
- [8] K. Horio and H. Yanai, “Numerical Modeling of Heterojunctions Including the Thermionic Emission Mechanism at the Heterojunction Interface,” *IEEE Transactions on Electron Devices*, vol. 37, no. 4, pp. 1093–1098, 1990.

19: Heterostructure Device Simulation

References

This chapter describes extensions for the temperature-dependent models.

Overview

Sentaurus Device provides the possibility to specify some parameters as a ratio of two irrational polynomials. The general form of such ratio is written as:

$$G(w, s) = f \frac{\left(\left(\sum a_i w^{p_i}\right) + d_n s\right)^{g_n}}{\left(\left(\sum a_j w^{p_j}\right) + d_d s\right)^{g_d}} \quad (584)$$

where subscripts n and d corresponds to numerator and denominator, respectively; f is a factor, w is a primary variable, and s is an additional variable. It is possible to use Eq. 584 with different coefficients for different intervals k defined by the segment $[w_{k-1}^{\max}, w_k^{\max}]$. By default, it is assumed that only one interval $k = 0$ with the boundaries $[0, \infty]$ exists, and function G is constant, that is, $a_0 = 0$, $a_i = 0$, $p = d = 0$, $g = 1$. Factor f is defined accordingly for each model.

A simplified syntax is introduced to define the piecewise linear function G . The boundaries of the intervals and the value of factor must be specified, which means the value of G is at the right side of the interval. All other coefficients should not be specified to use this possibility. As there are some peculiarities in parameter specification and model activation, the specific models for which the approximation by Eq. 584 is supported are described here separately.

Energy-dependent Energy Relaxation Time

For the specification of the energy relaxation time, the following modification of Eq. 584 is used:

$$\tau(w) = \tau_w^0 \frac{\left(\left(\sum a_i w^{p_i}\right)\right)^{g_n}}{\left(\left(\sum a_j w^{p_j}\right)\right)^{g_d}} \quad (585)$$

20: Energy-dependent Parameters

Energy-dependent Energy Relaxation Time

where $w = 1.5kT_n/q$ for electrons and $w = kT_p/q$ for holes. The factor f in Eq. 584 is defined by τ_w^0 , which can be specified in the parameter file by the values `tau_w_ele` and `tau_w_hol`.

To activate the specification of the energy-dependent energy relaxation time, the parameter `Formula(tau_w_ele)` (or `Formula(tau_w_hol)` for holes) must be set to 2.

The following example shows the energy relaxation time section of the parameter file and provides a short description of the syntax:

```
EnergyRelaxationTime
{ * Energy relaxation times in picoseconds
  tau_w_ele = 0.3 # [ps]
  tau_w_hol = 0.25 # [ps]
  * Below is the example of energy relaxation time approximation
  * by the ratio of two irrational polynomials.
  * If Wmax(interval-1) < Wc < Wmax(interval), then:
  * tau_w = (tau_w)*(Numerator^Gn)/(Denominator^Gd),
  * where (Numerator or Denominator)=SIGMA[A(i)(Wc^P(i))],
  * Wc=1.5(k*Tcar)/q (in eV).
  * By default: Wmin(0)=Wmax(-1)=0; Wmax(0)=infinity.
  * The option can be activated by specifying appropriate Formula equals 2
  *   Formula(tau_w_ele) = 2
  *   Formula(tau_w_hol) = 2
  *   Wmax(interval)_ele =
  *   tau_w_ele(interval) =
  *   Numerator(interval)_ele{
    *     A(0) =
    *     P(0) =
    *     A(1) =
    *     P(1) =
    *     D =
    *     G =
  *   }
  *   Denominator(interval)_ele{
    *     A(0) =
    *     P(0) =
    *     D =
    *     G =
  *   }
  *   Wmax(interval)_hol =
  *   tau_w_hol(interval) =
  tau_w_ele = 0.3 # [ps]
  tau_w_hol = 0.25 # [ps]

  Formula(tau_w_ele) = 2
  Numerator(0)_ele{
    A(0) = 0.048200
```

```

P(0) = 0.00
A(1) = 1.00
P(1) = 3.500
A(2) = 0.0500
P(2) = 2.500
A(3) = 0.0018100
P(3) = 1.00
}
Denominator(0)_ele{
    A(0) = 0.048200
    P(0) = 0.00
    A(1) = 1.00
    P(1) = 3.500
    A(2) = 0.100
    P(2) = 2.500
}

```

The following example shows a simplified syntax for piecewise linear specification of energy relaxation time:

```

EnergyRelaxationTime:
{ * Energy relaxation times in picoseconds
  Formula(tau_w_ele) = 2
  tau_w_ele = 0.3      # [ps]

  Wmax(0)_ele = 0.5    # [eV]
  tau_w_ele(1) = 0.46   # [ps]

  Wmax(1)_ele = 1.0    # [eV]
  tau_w_ele(2) = 0.4    # [ps]

  Wmax(2)_ele = 2.0    # [eV]
  tau_w_ele(3) = 0.2    # [ps]

  tau_w_hol = 0.25      # [ps]
}

```

Spline Interpolation

Sentaurus Device also allows spline approximation of energy relaxation time over energy. In this case, the parameter `Formula(tau_w_ele)` for electron energy relaxation time (and similarly, parameter `Formula(tau_w_hol)` for hole energy relaxation time) must be equal to 3.

20: Energy-dependent Parameters

Energy-dependent Mobility

Inside the braces following the keyword `Spline(tau_w_ele)` (or `Spline(tau_w_hol)`), an energy [eV] and tau [ps] value pair must be specified in each line. For the values outside of the specified intervals, energy relaxation time is treated as a constant and equal to the closest boundary value.

The following example shows a spline approximation specification for energy-dependent energy relaxation time for electrons:

```
EnergyRelaxationTime {
    Formula(tau_w_ele) = 3
    Spline(tau_w_ele) {
        0.      0.3    # [eV] [ps]
        0.5     0.46   # [eV] [ps]
        1.      0.4    # [eV] [ps]
        2.      0.2    # [eV] [ps]
    }
}
```

NOTE Energy relaxation times can be either energy-dependent or mole fraction-dependent (see [Composition-dependent Models on page 551](#)), but not both.

Energy-dependent Mobility

In addition to the existing energy-dependent mobility models (such as Caughey–Thomas, where the effective field is computed inside Sentaurus Device as a function of the carrier temperature), a more complex, user-supplied mobility model can be defined. For such specification of energy-dependent mobility, a modification to [Eq. 584](#) is used:

$$\mu(\bar{w}, N_{\text{tot}}) = \mu_{\text{low}} \frac{\left(\left(\sum a_i \bar{w}^{p_i} \right) + d_n N_{\text{tot}} \right)^{g_n}}{\left(\left(\sum a_j \bar{w}^{p_j} \right) + d_d N_{\text{tot}} \right)^{g_d}} \quad (586)$$

where $\bar{w} = T_n/T$ for electrons or $\bar{w} = T_p/T$ for holes, and μ_{low} is the low field mobility.

To activate the model, `CarrierTemperatureDrivePolynomial`, the driving force keyword, must be specified as a parameter of the high-field saturation mobility model. Parameters of the polynomials must be defined in the `HydroHighFieldMobility` parameter set.

This example shows the output of the HydroHighFieldMobility section and the specification of coefficients:

```

HydroHighFieldDependence:
{ * Parameter specifications for the high field degradation in
  * some hydrodynamic models.
  * B) Approximation by the ratio of two irrational polynomials
  * (driving force 'CarrierTempDrivePolynomial'):
  * If Wmax(interval-1) < w < Wmax(interval), then:
  * mu_hf = mu*factor*(Numerator^Gn)/(Denominator^Gd),
  * where (Numerator or Denominator)={SIGMA[A(i)(w^P(i))] + D*Ni},
  * w=Tc/Tl; Ni(cm^-3) is total doping.
  * By default: Wmin(0)=Wmax(-1)=0; Wmax(0)=infinity.

  *      Wmax(interval)_ele =
  *      F(interval)_ele =
  *      Numerator(interval)_ele{
  *          A(0)   =
  *          P(0)   =
  *          A(1)   =
  *          P(1)   =
  *          D     =
  *          G     =
  *      }
  *      Denominator(interval)_ele{
  *          A(0)   =
  *          P(0)   =
  *          D     =
  *          G     =
  *      }
  *      F(interval)_hol =
  *      Wmax(interval)_hol =
  Denominator(0)_ele
{
  A(0)   = 0.3
  P(0)   = 0.0
  A(1)   = 1.0
  P(1)   = 2.
  A(2)   = 0.001
  P(2)   = 2.500
  D     = 3.00e-16
  G     = 0.2500
}
}

```

20: Energy-dependent Parameters

Energy-dependent Mobility

Spline Interpolation

Instead of the rational polynomial in [Eq. 586](#), a spline interpolation can be used as well. In this case, the option `CarrierTempDriveSpline` must be used for the high-field mobility model in the command file:

```
Physics {
    Mobility (
        HighFieldSaturation (CarrierTempDriveSpline)
    )
}
```

The energy-dependent mobility is computed as

$$\mu(\bar{w}) = \mu_{\text{low}} \cdot \text{spline}(\bar{w}) \quad (587)$$

where the function `spline(\bar{w})` is defined by a sequence of value pairs in the parameter file:

```
HydroHighFieldDependence {
    Spline (electron) {
        0   1
        1   1
        2   2.5
        4   4
        10  5
    }

    Spline (hole) {
        0   1
        1   1
        2   0.75
        4   0.5
        10  0.2
    }
}
```

The given data points are interpolated by a cubic spline. Zero derivatives are imposed as boundary conditions at the end points. The spline function remains constant beyond the end points.

Energy-dependent Peltier Coefficient

Sentaurus Device allows for the following modification of the expression of the energy flux equation:

$$\vec{S}_n = -\frac{5r_n}{2} \left(\frac{kT_n}{q} \vec{J}_n + f_n^{\text{hf}} \kappa_n \frac{\partial(w_n \Pi_n)}{\partial w_n} \nabla T_n \right) \quad (588)$$

The standard expression corresponds to $\Pi_n = 1$. If $\Pi_n = 1 + P(\bar{w})$, then:

$$\frac{\partial(w_n \Pi_n)}{\partial w_n} = 1 + \bar{w} \frac{\partial P(\bar{w})C}{\partial \bar{w}} \quad (589)$$

Sentaurus Device allows you to specify the function Q :

$$Q(\bar{w}) = \bar{w} \frac{\partial P(\bar{w})}{\partial \bar{w}} \quad (590)$$

For the specification of Q , the following modification of Eq. 584 is used:

$$Q(\bar{w}) = f \frac{\left(\left(\sum a_i \bar{w}^{p_i} \right) \right)^{g_n}}{\left(\left(\sum a_j \bar{w}^{p_j} \right) \right)^{g_d}} \quad (591)$$

Coefficients must be specified in the HeatFlux parameter set, and the dependence can be activated by specifying a nonzero factor f .

For $\Pi_n = 1 + 1/(\sqrt{1 + \bar{w}^2})$, the result is $Q = 1/(\bar{w}^2 + 1)^{1.5}$. This is an example of the parameter file section for such a function Q_n :

```
HeatFlux
{ * Heat flux factor (0 <= hf <= 1)
  hf_n = 1    # [1]
  hf_p = 1    # [1]
  * Coefficients can be defined also as:
  *   hf_new = hf*(1.+Delta(w))
  * where Delta(w) is the ratio of two irrational polynomials.
  * If Wmax(interval-1) < Wc < Wmax(interval), then:
  * Delta(w) = factor*(Numerator^Gn)/(Denominator^Gd),
  * where (Numerator or Denominator)=SIGMA[A(i)(w^P(i))], w=Tc/Tl
  * By default: Wmin(0)=Wmax(-1)=0; Wmax(0)=infinity.
  * Option can be activated by specifying nonzero 'factor'.
  *   Wmax(interval)_ele =
  *   F(interval)_ele = 1
  *   Numerator(interval)_ele{
```

20: Energy-dependent Parameters

Energy-dependent Peltier Coefficient

```
*      A(0)  =
*      P(0)  =
*      A(1)  =
*      P(1)  =
*      G     =
*
*      }
*      Denominator(interval)_ele{
*          A(0)  =
*          P(0)  =
*          G     =
*
*          }
*          Wmax(interval)_hol =
*          F(interval)_hol = 1
*          f(0)_ele = 1
*          Denominator(0)_ele{
*              A(0)  = 1.
*              P(0)  = 0.
*              A(1)  = 1.
*              P(1)  = 2.
*              G     = 1.5
*          }
```

Spline Interpolation

Instead of the rational polynomial in Eq. 591, a spline interpolation can be used as well. The function $Q(\bar{w})$ is defined in the parameter file by a sequence of value pairs:

```
HeatFlux {
    hf_n = 1
    hf_p = 1

    Spline (electron) {
        0   1
        1   1
        2   2.5
        4   4
        10  5
    }

    Spline (hole) {
        0   1
        1   1
        2   0.75
        4   0.5
        10  0.2
    }
}
```

The given data points are interpolated by a cubic spline. Zero derivatives are imposed as boundary conditions at the end points. The spline function remains constant beyond the end points.

20: Energy-dependent Parameters

Energy-dependent Peltier Coefficient

This chapter discusses the anisotropic properties of semiconductor devices.

Overview

In general, all equations for semiconductor devices can be written in the following form:

$$-\nabla \cdot \vec{J} = R \quad (592)$$

Here vector $\vec{J} = \mu A \vec{g}$ where μ is a scalar function, tensor A is the 3×3 (2×2 , in the 2D case) symmetric matrix, and vector \vec{g} is a first-order differential expression. In the isotropic case, tensor A is the unit matrix. There are two reasons why tensor A may be anisotropic: anisotropic properties of semiconductors (such as SiC) and mechanical stress (see [Chapter 23 on page 595](#)). Sentaurus Device has three approximations for solving an anisotropic task.

The first and default approximation uses local (mesh vertex) linear transformation, which transforms an anisotropic task to an isotropic case (as a result, the transformed mesh can be non-Delaunay). After this, the algorithm uses AverageBoxMethod to compute control volumes and coefficients (see [Chapter 33 on page 885](#)). The keyword AverageAniso in the Math section activates this algorithm. It is active by default and can be switched off by specifying -AverageAniso. Due to the requirements of AverageBoxMethod, the AverageAniso option requires that either the input mesh consists of triangles only (tetrahedra in 3D) or the mesh is tensorial, with the main axes of this tensor mesh and the main axes of the anisotropy aligned to the simulation coordinate system.

The second approximation (for stress tasks only) is simple and correct only for tensor grids or near-tensor grids. The anisotropic effects are modeled using a tensor-grid approximation, where off-diagonal elements of tensor A are ignored. The diagonal elements of the tensor A are used as multiplication factors for the projections of the vector \vec{g} on mesh edges. The keyword TensorGridAniso in the Math section activates this algorithm.

The third approximation (for anisotropic tasks only) corrects the isotropic discretization of the semiconductor equations and it is correct if the off-diagonal elements of the tensor A are small (in comparison to diagonal values). The keyword -AverageAniso in the Math section activates this algorithm.

NOTE If the mesh is a tensor grid and tensor A has no off-diagonal elements, all three approximations are correct and the solutions are the same.

21: Anisotropic Properties

Anisotropic Mobility

Anisotropic Mobility

In some semiconductors, such as silicon carbide, the electrons and holes may exhibit different mobilities along different crystallographic axes.

Crystal Reference System

The crystal reference system of the semiconductor can be specified in the parameter file as follows:

```
LatticeParameters {  
    X = (1, 0, 0)  
    Y = (0, 1, 0)  
}
```

Instead of `LatticeParameters`, the keywords `Piezo` and `PiezoParameters` are recognized as well. By default, Sentaurus Device uses `X=(1,0,0)`, `Y=(0,1,0)`, and `Z=(0,0,1)`.

Anisotropy Factor

In a 3D simulation, Sentaurus Device assumes that the electrons or holes exhibit a mobility μ along the x-axis and y-axis, and an anisotropic mobility μ_{aniso} along the z-axis. In a 2D simulation, the regular mobility μ is observed along the x-axis, and μ_{aniso} is observed along the y-axis. The anisotropy factor r is defined as the ratio:

$$r = \frac{\mu}{\mu_{\text{aniso}}} \quad (593)$$

Current Densities

In the isotropic case, the current densities can be expressed by:

$$\vec{J}_n = \mu_n \vec{g}_n \quad (594)$$

$$\vec{J}_p = \mu_p \vec{g}_p \quad (595)$$

where \vec{g}_n and \vec{g}_p are the *currents without mobilities*. In the drift-diffusion model, we have:

$$\vec{g}_n = -nq \nabla \Phi_n \quad (596)$$

$$\vec{g}_p = -pq \nabla \Phi_p \quad (597)$$

as can be seen from [Eq. 28, p. 179](#) and [Eq. 29, p. 179](#). For the thermodynamic model, [Eq. 30, p. 179](#) and [Eq. 31, p. 179](#) imply that:

$$\vec{g}_n = -nq(\nabla \Phi_n + P_n \nabla T) \quad (598)$$

$$\vec{g}_p = -pq(\nabla \Phi_p + P_p \nabla T) \quad (599)$$

In the hydrodynamic case, we have:

$$\vec{g}_n = n \nabla E_C + kT_n \nabla n + f_n^{\text{td}} kn \nabla T_n - 1.5nkT_n \nabla \ln m_n \quad (600)$$

$$\vec{g}_p = p \nabla E_V - kT_p \nabla p - f_p^{\text{td}} kp \nabla T_p - 1.5pkT_p \nabla \ln m_p \quad (601)$$

according to [Eq. 36, p. 185](#) and [Eq. 37, p. 185](#).

For anisotropic mobilities, [Eq. 594](#) and [Eq. 595](#) need to be rewritten as:

$$\vec{J}_n = \mu_n A_{r_n} \vec{g}_n \quad (602)$$

$$\vec{J}_p = \mu_p A_{r_p} \vec{g}_p \quad (603)$$

21: Anisotropic Properties

Anisotropic Mobility

where r_n and r_p are the anisotropy factors for electrons and holes, respectively. If the crystal reference system coincides with the coordinate system of Sentaurus Device, the matrices A_r are given by:

$$A_r = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1/r \end{bmatrix} \text{ or } A_r = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1/r \end{bmatrix} \quad (604)$$

depending on the dimension of the problem.

In general, however, A_r needs to be written as:

$$A_r = Q \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1/r \end{bmatrix} Q^T \text{ or } A_r = Q_{2:2} \begin{bmatrix} 1 & \\ & 1/r \end{bmatrix} Q_{2:2}^T \quad (605)$$

where Q is the 3×3 orthogonal matrix:

$$Q = \begin{bmatrix} \vec{x} & \vec{y} & \vec{z} \\ \| \vec{x} \|_2 & \| \vec{y} \|_2 & \| \vec{z} \|_2 \end{bmatrix} \quad (606)$$

and the quantity $Q_{2:2}$ denotes the leading 2×2 submatrix of Q . For 2D problems, the vectors \vec{x} and \vec{y} must lie in the xy plane.

Therefore, Q will have the form:

$$Q = \begin{bmatrix} Q_{2:2} & 0 \\ 0 & \pm 1 \end{bmatrix} \quad (607)$$

Driving Forces

In the isotropic case, the electric field parallel to the electron or hole current is given by (see Eq. 270):

$$F_c = \frac{\vec{F} \cdot \vec{J}_c}{J_c} = \frac{\vec{F} \cdot \vec{g}_c}{g_c} \quad (608)$$

For anisotropic mobilities:

$$F_c = \frac{\vec{F} \cdot A \vec{g}_c}{|A \vec{g}_c|} \quad (609)$$

Similarly, the electric field perpendicular to the current, as given in [Eq. 241, p. 287](#), needs to be rewritten as:

$$F_{c,\perp} = \sqrt{F^2 - \frac{(\vec{F} \cdot A\vec{g}_c)^2}{|A\vec{g}_c|^2}} \quad (610)$$

[Eq. 271, p. 300](#) shows how the gradient of the Fermi potential Φ_c may be used as the driving force in high-field saturation models. Instead, for anisotropic mobilities, Sentaurus Device uses:

$$F_c = A\nabla\Phi_c \quad (611)$$

In the isotropic hydrodynamic Canali model, the driving force \vec{F}_c satisfies:

$$\vec{F}_c \cdot \mu\vec{F}_c = \frac{w_c - w_0}{\tau_{ec}q} \quad (612)$$

as can be seen from [Eq. 273, p. 301](#). To derive the appropriate expression in the anisotropic case we assume that \vec{F}_c operates parallel to the current, that is:

$$\vec{F}_c = F_c \hat{e}_c \quad (613)$$

where:

$$\hat{e}_c = \frac{A\vec{g}_c}{|A\vec{g}_c|} \quad (614)$$

is the direction of the electron or hole current. Instead of [Eq. 612](#), we now have:

$$\mu F_c^2 (A\hat{e}_c) \cdot \hat{e}_c = \frac{w_c - w_0}{\tau_{ec}q} \quad (615)$$

or:

$$F_c = \sqrt{\frac{w_c - w_0}{\tau_{ec}q\mu A\hat{e}_c \cdot \hat{e}_c}} \quad (616)$$

21: Anisotropic Properties

Anisotropic Mobility

Total Anisotropic Mobility

This is the simplest mode in Sentaurus Device. Only a total anisotropy factor r_e or r_h is specified in the command file:

```
Physics {  
    Aniso(  
        eMobilityFactor (Total) = re  
        hMobilityFactor (Total) = rh  
    )  
}
```

Sentaurus Device computes the mobility μ for electrons or holes along the main crystallographic axis as usual. The mobility μ_{aniso} is then given by:

$$\mu_{\text{aniso}} = \frac{\mu}{r} \quad (617)$$

NOTE In this mode, Sentaurus Device does not update the driving forces as discussed in [Driving Forces on page 578](#).

Total Direction-dependent Anisotropic Mobility

This mode is activated by specifying the total direction-dependent anisotropy factor r_e or r_h in the command file of Sentaurus Device:

```
Physics {  
    Aniso(  
        eMobilityFactor (TotalDD) = re  
        hMobilityFactor (TotalDD) = rh  
    )  
}
```

First, Sentaurus Device updates the driving forces as discussed in [Driving Forces on page 578](#). Afterwards, the electron or hole mobility μ along the main crystallographic axis is computed as usual, and the mobility μ_{aniso} is given by:

$$\mu_{\text{aniso}} = \frac{\mu}{r} \quad (618)$$

Self-Consistent Anisotropic Mobility

This is the most accurate, but also the most expensive, mode in Sentaurus Device. The electron and hole mobility models specified in the Physics section are evaluated separately for the major and minor crystallographic axes, but with different parameters for each axis. This option is activated in the Physics section for electron or hole mobilities as follows:

```
Physics {
    Aniso(
        eMobility
        hMobility
    )
}
```

To simplify matters, it is possible to specify:

```
Physics {
    Aniso(
        Mobility
    )
}
```

to activate self-consistent, anisotropic, mobility calculations for both electrons and holes. [Table 92](#) lists the mobility models that offer an anisotropic version.

Table 92 Anisotropic mobility models

Isotropic model	Anisotropic model
ConstantMobility	ConstantMobility_aniso
DopingDependence	DopingDependence_aniso
EnormalDependence	EnormalDependence_aniso
HighFieldDependence	HighFieldDependence_aniso
UniBoDopingDependence	UniBoDopingDependence_aniso
UniBoEnormalDependence	UniBoEnormalDependence_aniso
UniBoHighFieldDependence	UniBoHighFieldDependence_aniso
HydroHighFieldDependence	HydroHighFieldDependence_aniso

The PMI also supports anisotropic mobility calculations. The constructors of the classes `PMI_DopingDepMobility`, `PMI_EnormalMobility`, and `PMI_HighFieldMobility` contain an additional flag to distinguish between the isotropic and anisotropic case (see [Chapter 34 on page 911](#)).

21: Anisotropic Properties

Anisotropic Mobility

For example, you can specify these parameters for the constant mobility model in the parameter file of Sentaurus Device:

```
ConstantMobility {  
    mumax = 1.4170e+03, 4.7050e+02  
    Exponent = 2.5, 2.2  
}
```

The following parameters would then compute a reduced constant mobility along the anisotropic axis:

```
ConstantMobility_aniso {  
    mumax = 1.0e+03, 4.0e+02  
    Exponent = 2.5, 2.2  
}
```

In each vertex, Sentaurus Device introduces the anisotropy factors r_e and r_h as two additional unknowns. For a given value of r , the driving forces F_c and $F_{c,\perp}$ are computed as discussed in [Driving Forces on page 578](#), and the mobilities along the isotropic and anisotropic axes are obtained.

The equation for the unknown factor r is then given by:

$$r = \frac{\mu(r)}{\mu_{\text{aniso}}(r)} \quad (619)$$

This nonlinear equation is solved in each vertex for both electron and hole mobilities.

Plot Section

[Table 93](#) lists the plot variables that may be useful for visualizing anisotropic mobility calculations.

Table 93 Plot variables for anisotropic mobility

Plot variable	Description
eMobility	Electron mobility along main axis
hMobility	Hole mobility along main axis
eMobilityAniso	Electron mobility along anisotropic axis
hMobilityAniso	Hole mobility along anisotropic axis
eMobilityAnisoFactor	Anisotropic factor for electrons
hMobilityAnisoFactor	Anisotropic factor for holes

Anisotropic Avalanche Generation

Sentaurus Device computes the avalanche generation according to [Eq. 316, p. 324](#). In the isotropic case, the terms $n v_n$ and $p v_p$ can also be written, respectively, as:

$$n v_n = \mu_n |\vec{g}_n| \quad (620)$$

and:

$$p v_p = \mu_p |\vec{g}_p| \quad (621)$$

If anisotropic mobilities are switched on, [Eq. 620](#) and [Eq. 621](#) are replaced by:

$$n v_n = \mu_n |A_{r_n} \vec{g}_n| \quad (622)$$

and:

$$p v_p = \mu_p |A_{r_p} \vec{g}_p| \quad (623)$$

NOTE [Eq. 622](#) and [Eq. 623](#) only apply to total direction-dependent and self-consistent mobility calculations. If the total anisotropic option (see [Total Anisotropic Mobility on page 580](#)) is selected, [Eq. 620](#) and [Eq. 621](#) are used.

Anisotropic avalanche calculations can be activated in the `Physics` section, independently for electrons and holes:

```
Physics {
    Aniso(
        eAvalanche
        hAvalanche
    )
}
```

The keyword `Avalanche` activates calculations of anisotropic avalanche for both electrons and holes:

```
Physics {
    Aniso(
        Avalanche
    )
}
```

21: Anisotropic Properties

Anisotropic Electrical Permittivity

In the anisotropic mode, different avalanche parameters can be specified along the isotropic and anisotropic axes. [Table 94](#) shows the avalanche models that are supported.

Table 94 Anisotropic avalanche models

Isotropic model	Anisotropic model
vanOverstraetendeMan	vanOverstraetendeMan_aniso
Okuto	Okuto_aniso

Sentaurus Device uses interpolation to compute avalanche parameters for an arbitrary direction of the current. Let \hat{e}_c be the direction of the electron or hole current as defined in [Eq. 614](#). In the crystal reference system, the current is given by:

$$\hat{e}'_c = Q^T \hat{e}_c = \begin{bmatrix} e'_x \\ e'_y \\ e'_z \end{bmatrix} \quad (624)$$

Sentaurus Device interpolates an avalanche parameter p depending on the direction of the current \hat{e}'_c according to:

$$p(\hat{e}'_c) = (e'^2_x + e'^2_y) \cdot p_{\text{isotropic}} + e'^2_z \cdot p_{\text{anisotropic}} \quad (625)$$

in a 3D simulation, and, in a 2D simulation:

$$p(\hat{e}'_c) = e'^2_x \cdot p_{\text{isotropic}} + e'^2_y \cdot p_{\text{anisotropic}} \quad (626)$$

The PMI also supports the calculation of anisotropic avalanche generation. The current without mobility $A\vec{g}_c$ is passed as an input parameter, and it can be used by the PMI code to determine the model parameters depending on the direction of the current (see [Avalanche Generation Model on page 935](#)).

Anisotropic Electrical Permittivity

The electrical permittivity ϵ in [Eq. 26, p. 178](#) can have different values along different crystallographic axes. If the crystallographic axes coincide with the coordinate system of Sentaurus Device, the scalar ϵ is replaced by the matrix:

$$E = \begin{bmatrix} \epsilon & & \\ & \epsilon & \\ & & \epsilon_{\text{aniso}} \end{bmatrix} \text{ or } E = \begin{bmatrix} \epsilon & \\ & \epsilon_{\text{aniso}} \end{bmatrix} \quad (627)$$

depending on the dimension of the problem. For general crystallographic axes, the matrix E is given by:

$$E = Q \begin{bmatrix} \varepsilon & & \\ & \varepsilon & \\ & & \varepsilon_{\text{aniso}} \end{bmatrix} Q^T \quad \text{or} \quad E = Q_{2:2} \begin{bmatrix} \varepsilon & \\ & \varepsilon_{\text{aniso}} \end{bmatrix} Q_{2:2}^T \quad (628)$$

where Q is defined in [Eq. 606](#).

Anisotropic electrical permittivity is switched on by using the keyword `Poisson` in the `Physics` section of the command file:

```
Physics {
    Aniso (Poisson)
}
```

NOTE This command should only appear in the global `Physics` section. Regionwise activation of anisotropic electrical permittivity is not supported.

The model parameters for ε and $\varepsilon_{\text{aniso}}$ can be specified in the parameter file. [Table 95](#) lists the names of the corresponding models.

Table 95 Anisotropic electrical permittivity models

Isotropic model	Anisotropic model
Epsilon	Epsilon_aniso

Different parameters can be specified for each region or each material. The following statement in the command file of Sentaurus Device can be used to plot the electrical permittivities:

```
Plot {
    DielectricConstant
    "DielectricConstantAniso"
}
```

Anisotropic Thermal Conductivity

The thermal conductivity κ in [Eq. 32, p. 180](#) can have different values along different crystallographic axes. If the crystallographic axes coincide with the coordinate system of Sentaurus Device, the scalar κ is replaced by the matrix:

$$K = \begin{bmatrix} \kappa & & \\ & \kappa & \\ & & \kappa_{\text{aniso}} \end{bmatrix} \text{ or } K = \begin{bmatrix} \kappa & & \\ & & \kappa_{\text{aniso}} \end{bmatrix} \quad (629)$$

depending on the dimension of the problem.

For general crystallographic axes, the matrix K is given by:

$$K = Q \begin{bmatrix} \kappa & & \\ & \kappa & \\ & & \kappa_{\text{aniso}} \end{bmatrix} Q^T \text{ or } K = Q_{2:2} \begin{bmatrix} \kappa & & \\ & & \kappa_{\text{aniso}} \end{bmatrix} Q_{2:2}^T \quad (630)$$

where Q is defined in [Eq. 606](#).

Anisotropic thermal conductivity is switched on by the keyword Temperature in the Physics section of the command file:

```
Physics {
    Aniso (Temperature)
}
```

This command should only appear in the global Physics section. Regionwise activation of anisotropic thermal conductivity is not supported.

The model parameters for κ and κ_{aniso} can be specified in the parameter file. [Table 96](#) lists the names of the corresponding models.

Table 96 Anisotropic thermal conductivity models

Isotropic model	Anisotropic model
Kappa	Kappa_aniso

Different parameters can be specified for each region or each material. The PMI can also be used to compute anisotropic thermal conductivities. The constructor of the class PMI_ThermalConductivity has an additional parameter to distinguish between the isotropic and anisotropic directions (see [Thermal Conductivity on page 989](#)).

The following statement in the command file can be used to plot the thermal conductivities:

```
Plot {  
    "ThermalConductivity"  
    "ThermalConductivityAniso"  
}
```

Anisotropic Density Gradient Model

The density gradient model provides support for anisotropic quantization. For more details, see [Density Gradient Quantization Model on page 260](#).

21: Anisotropic Properties

Anisotropic Density Gradient Model

CHAPTER 22 Ferroelectric Materials

This chapter explains how ferroelectric materials are treated in a simulation using Sentaurus Device.

In ferroelectric materials, the polarization \vec{P} depends nonlinearly on the electric field \vec{F} . The polarization at a given time depends on the electric field at that time and the electric field at previous times. The history dependence leads to the well-known phenomenon of hysteresis, which is used in nonvolatile memory technology.

Using Ferroelectrics

Sentaurus Device implements a model for ferroelectrics that features minor loop nesting and memory wipeout. [Figure 57](#) demonstrates these properties and [Ferroelectrics Model on page 591](#) discusses them further.

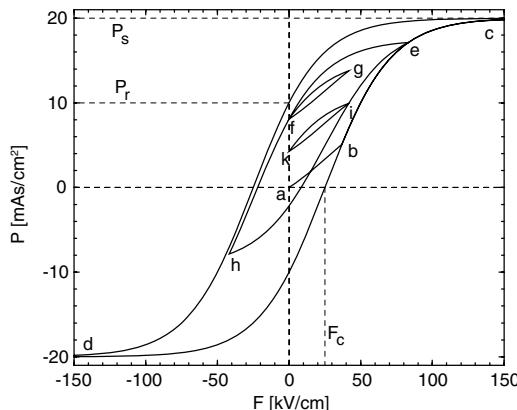


Figure 57 Example polarization curve

To activate the model, specify the keyword `Polarization` in the `Physics` section of the command file. Use the optional parameter `Memory` to prescribe the maximum allowed nesting depth of minor loops. The smallest allowed value for `Memory` is 2; the default value is 10. If minor loop nesting becomes too deep, the nesting property of the minor loops can be lost. However, the polarization curve remains continuous. For example:

```
Physics (region = "Region.17") {  
    Polarization (Memory=20)  
}
```

22: Ferroelectric Materials

Using Ferroelectrics

switches on the ferroelectric model in region Region.17 and sets the size of the memory to 20 turning points for each element and each mesh axis.

To obtain a plot of the polarization field, specify `Polarization/Vector` in the `Plot` section of the command file.

Sentaurus Device characterizes the static properties of a ferroelectric material by three parameters: the remanent polarization P_r , the saturation polarization P_s , and the coercive field F_c . The hysteresis curve in [Figure 57 on page 589](#) illustrates these quantities. Furthermore, Sentaurus Device parameterizes the transient response of the ferroelectric material by the relaxation times τ_E and τ_P , and by a nonlinear coupling constant k_n (see [Ferroelectrics Model on page 591](#)).

Specify the values for these parameters in the `Polarization` parameter set, for example:

```
Polarization
{
    * Remanent polarization P_r, saturation polarization P_s,
    * and coercive field F_c for x,y,z direction (crystal axes)
        P_r = (1.0000e-05, 1.0000e-05, 1.0000e-05) #[C/cm^2]
        P_s = (2.0000e-05, 2.0000e-05, 2.0000e-05) #[C/cm^2]
        F_c = (2.5000e+04, 2.5000e+04, 2.5000e+04) #[V/cm]
    * Relaxation time for the auxiliary field tau_E, relaxation
    * time for the polarization tau_P, nonlinear coupling kn.
        tau_E = (0.0000e+00, 0.0000e+00, 0.0000e+00) #[s]
        tau_P = (0.0000e+00, 0.0000e+00, 0.0000e+00) #[s]
        kn      = (0.0000e+00, 0.0000e+00, 0.0000e+00) #[cm*s/V]
}
```

The parameters in this example are the defaults for the material `InsulatorX`. For all other materials, all default values are zero. Each of the three numbers given for any of the parameters corresponds to the value for the respective coordinate axis of the mesh. If a P_s component is zero, the ferroelectric model is disabled along the corresponding direction. If a P_s component is nonzero, the respective P_r and F_c components must also be nonzero. Furthermore, the P_r component must be smaller than the P_s component. By default, the relaxation times are zero, which means that polarization follows the applied electric field instantaneously.

In devices with ferroelectric and semiconductor regions, it is sometimes difficult to obtain an initial solution of the Poisson equation. In many cases, the `LineSearchDamping` option can solve these problems. To use this option, start the simulation like:

```
coupled (LineSearchDamping=0.01) { Poisson }
```

See [Coupled Command on page 65](#) for details about this parameter.

Ferroelectrics Model

The vector quantity \vec{P} is split into its components along the main axes of the mesh coordinate system. This results in one to three scalar problems. Sentaurus Device handles each problem separately using the model from [1] with extensions for transient behavior [2].

First, Sentaurus Device computes an auxiliary field F_{aux} from the electric field F :

$$\frac{d}{dt} F_{\text{aux}}(t) = \frac{F(t) - F_{\text{aux}}(t)}{\tau_E} \quad (631)$$

Here, τ_E is a material-specific time constant. For $\tau_E = 0$ or for quasistationary simulations, $F_{\text{aux}} = F$.

From the auxiliary field, Sentaurus Device computes the auxiliary polarization P_{aux} . The auxiliary polarization P_{aux} is an algebraic function of the auxiliary field F_{aux} :

$$P_{\text{aux}} = c \cdot P_s \cdot \tanh(w \cdot (F_{\text{aux}} \pm F_c)) + P_{\text{off}} \quad (632)$$

where P_s is the saturation polarization, F_c is the coercive field, and:

$$w = \frac{1}{2F_c} \ln \frac{P_s + P_r}{P_s - P_r} \quad (633)$$

where P_r is the remanent polarization. In Eq. 632, the plus sign applies to the decreasing auxiliary field and the minus sign applies to the increasing auxiliary field. The different signs reflect the hysteretic behavior of the material. P_{off} and c in Eq. 632 result from the polarization history of the material, see below.

Finally, from the auxiliary polarization and auxiliary field, Sentaurus Device computes the actual polarization P :

$$\frac{d}{dt} P(t) = \frac{P_{\text{aux}}[F_{\text{aux}}(t)] - P(t)}{\tau_P} \left(1 + k_n \left| \frac{d}{dt} F_{\text{aux}}(t) \right| \right) \quad (634)$$

Here, τ_P and k_n are material-specific constants. For $\tau_P = 0$ or for quasistationary simulations, $P = P_{\text{aux}}$.

Upper and lower turning points are points in the $P_{\text{aux}} - F_{\text{aux}}$ diagram where the sweep direction of the auxiliary field F_{aux} changes from increasing to decreasing, and from decreasing to increasing, respectively. At each bias point, the most recent upper and lower turning points, (F_u, P_u) and (F_l, P_l) , must both be on each of the two curves defined by Eq. 632; this requirement determines P_{off} and c .

22: Ferroelectric Materials

Ferroelectrics Model

Sentaurus Device ‘memorizes’ turning points as they are encountered during a simulation. The memory always contains (∞, P_s) as the oldest and $(-\infty, -P_s)$ as the second oldest turning point. By using Eq. 632, these two points define a pair of curves with $c = 1$ and $P_{\text{off}} = 0$; together, the two curves form the saturation loop. All other pairs of turning points result in $c < 1$ and define a pair of curves forming minor loops.

When the auxiliary field leaves the interval defined by F_l and F_u of the two newest turning points, these two turning points are removed from the memory; this reflects the memory wipeout observed in experiments. The pair of turning points that are newest in the memory, after this removal, determines the further $P_{\text{aux}}(F_{\text{aux}})$ relationship.

As the older of the dropped turning points was originally reached by walking on the curve defined by the turning points that now (after dropping) again determine $P_{\text{aux}}(F_{\text{aux}})$, the polarization curve remains continuous. For example, see points e, f, and i in Figure 57 on page 589. Furthermore, the minor loop defined by the two dropped turning points is nested inside the minor loop defined by the present turning points. The nesting of minor loops is also a feature known from experiments on ferroelectrics. Figure 57 illustrates this by the loops f-g and i-k, both of which are nested in loop e-h, which in turn is nested in loop c-d.

In small-signal (AC) analysis (see ACCoupled: Small-Signal AC Analysis on page 161), a very small periodic signal is added to the DC bias. As a result, the (auxiliary) polarization at each point of the ferroelectric material changes along a very small minor loop nested in the main loop that stems from the DC variation of the bias voltage. The average slope of this minor loop is always smaller than the slope of the main loop at the point where the loops touch. Consequently, even at very low frequencies, the AC response of the system is different from what would be obtained by taking the derivative of the DC curves.

As an example of how the turning point memory works, see Figure 57. The points on the polarization curve are reached in the sequence a, b, c, d, e, f, g, f, h, i, k, i, e, c. For simplicity, a quasistationary process is assumed and, therefore, $F_{\text{aux}} = F$ and $P_{\text{aux}} = P$. Starting the simulation at point a, the newest point in memory is b (Sentaurus Device initializes it in this way). The second newest point is a negative saturation point (l), and the oldest point is a positive saturation point (u). This memory state is denoted by [blu]. Therefore, the curve segment (that is, the coefficients c and P_{off}) from a to b is determined by the points l and b.

When crossing b and proceeding to c, b is dropped from the memory. As l is a saturation point, it is retained. Therefore, the memory becomes [lu]. These two points determine the curve from b to c. Turning at c, c is added to the memory, giving [clu]. From c to d, use c and l; at d, the memory becomes [dclu]; at point e, [edclu]; at f, [fedclu]; at g, [gfedclu]. Passing through f, the two newest points, f and g, are dropped and the memory is [edclu]; at h, [hedclu]; at i, [ihedclu]; at k, [kihedclu]. At i again, i and k are dropped, giving [hedclu]; at e, e and h are dropped, giving [dclu].

At the beginning of a simulation, the memory contains one turning point chosen such that the point $E_{\text{aux}} = 0, P_{\text{aux}} = 0$ is on the minor loop so defined (for example, point b in [Figure 57 on page 589](#)). The nature of the model is such that it is not possible to have a state of the system that is completely symmetric. In particular, even for a symmetric device and at the very beginning of the simulation, $P(E) \neq -P(-E)$. This asymmetry is most prominent for the virgin curves (for example, a-b in [Figure 57](#)) of the ferroelectric, which are different for different signs of voltage ramping.

References

- [1] B. Jiang *et al.*, “Computationally Efficient Ferroelectric Capacitor Model for Circuit Simulation,” in *Symposium on VLSI Technology*, Kyoto, Japan, pp. 141–142, June 1997.
- [2] K. Dragosits, *Modeling and Simulation of Ferroelectric Devices*, Ph.D. thesis, Technische Universität Wien, Vienna, Austria, December 2000.

22: Ferroelectric Materials

References

This chapter presents an overview of the importance of stress in device simulation.

Stress engineering is a key point to ensuring the high performance of CMOS devices. Mechanical stress can affect workfunction, band gap, effective mass, carrier mobility, and leakage currents. The stress is generated by many technological processes due to different process temperatures and material properties. In addition, it can be added (such as silicon layers onto SiGe bulk) to improve device performance.

Overview

Mechanical distortion of semiconductor microstructures results in a change in the band structure and carrier mobility. This effect is well known and appropriate computations of the change in the strain-induced band structure are based on the deformation potential theory [1]. The implementation of the deformation potential model in Sentaurus Device is based on data and approaches presented in the literature [1][2][3][4]. Other approaches [5][6] implemented in Sentaurus Device focus more on the description of piezoresistive effects.

Generally, the stress tensor $\bar{\sigma}$ is a symmetric 3×3 matrix. Therefore, it only has six independent components.

With the index transformation rule, it can be written in a six-component vector notation $\sigma_{11} \rightarrow \sigma_1, \sigma_{22} \rightarrow \sigma_2, \sigma_{33} \rightarrow \sigma_3, \sigma_{23} \rightarrow \sigma_4, \sigma_{13} \rightarrow \sigma_5, \sigma_{12} \rightarrow \sigma_6$ that simplifies tensor expressions. For example, to compute the strain tensor ε (which is needed for the deformation potential model), the generalized Hooke's law for anisotropic materials is applied:

$$\varepsilon_i = \sum_{j=1}^6 S_{ij} \sigma_j \quad (635)$$

where S_{ij} is the elasticity modulus (see [Deformation of Band Structure on page 596](#)). Note that $\varepsilon_4, \varepsilon_5$, and ε_6 in Eq. 635 represent *engineering shear strain* components. The shear strain components that are used in model equations in the following sections are related to these by:

$$\begin{aligned} \varepsilon_{23} &= \varepsilon_{32} = \varepsilon_4/2 \\ \varepsilon_{31} &= \varepsilon_{13} = \varepsilon_5/2 \\ \varepsilon_{12} &= \varepsilon_{21} = \varepsilon_6/2 \end{aligned} \quad (636)$$

23: Modeling Mechanical Stress Effect

Deformation of Band Structure

In crystals with cubic symmetry such as silicon, the number of independent coefficients of the elasticity tensor (as other material property tensors) reduces to three by rotating the coordinate system parallel to the high-symmetric axes of the crystal [7]. This gives the following elasticity tensor \bar{S} :

$$\bar{S} = \begin{bmatrix} S_{11} & S_{12} & S_{12} & 0 & 0 & 0 \\ S_{12} & S_{11} & S_{12} & 0 & 0 & 0 \\ S_{12} & S_{12} & S_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & S_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & S_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 & S_{44} \end{bmatrix} \quad (637)$$

where the coefficients S_{11} , S_{12} , and S_{44} correspond to parallel, perpendicular, and shear components, respectively.

In Sentaurus Device, the stress tensor can be defined in the stress coordinate system $(\vec{e}_1, \vec{e}_2, \vec{e}_3)$. To transfer this tensor to another coordinate system (for example, the crystal system $(\vec{e}'_1, \vec{e}'_2, \vec{e}'_3)$, which is a common operation), the following transformation rule between two coordinate systems is applied:

$$\sigma'_{ij} = a_{ik}a_{jl}\sigma_{kl} \quad (638)$$

where \bar{a} is the rotation matrix:

$$a_{ik} = \frac{\vec{e}'_i \cdot \vec{e}_k}{|\vec{e}'_i||\vec{e}_k|} \quad (639)$$

Deformation of Band Structure

In deformation potential theory [2][8], the strains are considered to be relatively small. The change in energy of each carrier subvalley, caused by the deformation of the lattice, is a linear function of the strain. By default (for silicon), Sentaurus Device considers three bands for electrons (which are applied to three two-fold bands in the conduction band) and two bands for holes (which are applied to heavy-hole and light-hole bands in the valence band). The number of carrier bands can be changed in the parameter file (see [Using Deformation Potential Model on page 599](#)).

Bir and Pikus [8] proposed a model for the strain-induced change in the energy of carrier bands in silicon where they ignore the shear strain for electrons and suggest nonlinear dependence for holes:

$$\begin{aligned}\Delta E_{C,i} &= \Xi_d(\varepsilon'_{11} + \varepsilon'_{22} + \varepsilon'_{33}) + \Xi_u \varepsilon'_{ii} \\ \Delta E_{V,i} &= -a(\varepsilon'_{11} + \varepsilon'_{22} + \varepsilon'_{33}) \pm \delta E \\ \delta E &= \sqrt{\frac{b^2}{2}((\varepsilon'_{11} - \varepsilon'_{22})^2 + (\varepsilon'_{22} - \varepsilon'_{33})^2 + (\varepsilon'_{11} - \varepsilon'_{33})^2) + d^2(\varepsilon'_{12}^2 + \varepsilon'_{13}^2 + \varepsilon'_{23}^2)}\end{aligned}\quad (640)$$

where Ξ_d, Ξ_u, a, b, d are other deformation potentials that correspond to the model, i corresponds to the carrier band number, and ε'_{ij} are the components of the strain tensor in the crystal coordinate system (see [Overview on page 595](#) for a description of tensor transformations). The sign \pm separates heavy-hole and light-hole bands of silicon. Both expressions in [Eq. 640](#) have the common part $(\varepsilon'_{11} + \varepsilon'_{22} + \varepsilon'_{33})$ and, therefore, these expressions were combined in a general one that gives flexibility to its definition in the parameter file (see [Using Deformation Potential Model on page 599](#)):

$$\begin{aligned}\Delta E_{B,i} &= \xi_{i1}^{B2} \varepsilon'_{11} + \xi_{i2}^{B2} \varepsilon'_{22} + \xi_{i3}^{B2} \varepsilon'_{33} + \\ &\quad \xi_{i4}^{B2} \sqrt{\frac{(\xi_{i5}^{B2})^2}{2}((\varepsilon'_{11} - \varepsilon'_{22})^2 + (\varepsilon'_{22} - \varepsilon'_{33})^2 + (\varepsilon'_{11} - \varepsilon'_{33})^2) + (\xi_{i6}^{B2})^2(\varepsilon'_{12}^2 + \varepsilon'_{13}^2 + \varepsilon'_{23}^2)}\end{aligned}\quad (641)$$

where ξ_{ij}^{B2} are deformation potentials that correspond to the Bir and Pikus model, and ξ_{i4}^{B2} is a unitless constant that defines mainly a sign.

Sentaurus Device uses [Eq. 641](#) for all electron and hole bands by default, but there are other options that allow you to select more sophisticated models separately for electrons and holes. Using a degenerate $k \cdot p$ theory at the zone boundary X-point, the authors of [9] derived an additional shear term for the electron bands in [Eq. 640](#):

$$\Delta E_{C,i} = \Xi_d(\varepsilon'_{11} + \varepsilon'_{22} + \varepsilon'_{33}) + \Xi_u \varepsilon'_{ii} + \begin{cases} -\frac{\Delta}{4} \eta_i^2 & , |\eta_i| \leq 1 \\ -(2|\eta_i| - 1)\frac{\Delta}{4} & , |\eta_i| > 1 \end{cases} \quad (642)$$

where:

- $\eta_i = \frac{4\Xi_u \varepsilon'_{jk}}{\Delta}$ is a dimensionless off-diagonal strain with $j \neq k \neq i$.
- ε'_{jk} is a shear strain component.
- Δ is the band separation between the two lowest conduction bands.
- Ξ_u is the deformation potential responsible for the band-splitting of the two lowest conduction bands [9]: $(E_{\Delta_1} - E_{\Delta_2})|_{X[001]} = 4\Xi_u \varepsilon'_{jk}$.

23: Modeling Mechanical Stress Effect

Deformation of Band Structure

The strain-induced shifts of valence bands can be computed using 6x6 $k \cdot p$ theory for the heavy-hole, light-hole, and split-off bands as described in [10]. The specification of the deformation potentials for both of these models is described in [Using Deformation Potential Model on page 599](#).

Using the stress tensor $\bar{\sigma}$ from the input file, Sentaurus Device recomputes it from the stress coordinate system to the tensor $\bar{\epsilon}$ in the crystal system by Eq. 638. The strain tensor $\bar{\epsilon}$ is a result of applying Hooke's law Eq. 635 to the stress $\bar{\sigma}$. Using Eq. 641, Eq. 642, or solving the cubic equation from [10], the energy band change can be computed for each conduction and valence carrier bands. Sentaurus Device does not modify the effective masses, but it computes strain-induced conduction and valence band-edge shifts, ΔE_C and ΔE_V . By the default, an averaged value for the band-edge shifts is applied:

$$\begin{aligned}\frac{\Delta E_C}{kT_{300}} &= -\ln \left[\frac{1}{n_C} \sum_{i=1}^{n_C} \exp\left(\frac{-\Delta E_{C,i}}{kT_{300}}\right) \right] \\ \frac{\Delta E_V}{kT_{300}} &= \ln \left[\frac{1}{n_V} \sum_{i=1}^{n_V} \exp\left(\frac{\Delta E_{V,i}}{kT_{300}}\right) \right]\end{aligned}\quad (643)$$

where n_C and n_V are the number of subvalleys considered in the conduction and valence bands, respectively, and $T_{300} = 300\text{ K}$. In addition, there is a user-defined option that, instead of Eq. 643, applies lowest band energies for the conduction and valence band-edge shifts as follows:

$$\begin{aligned}\Delta E_C &= \min(\Delta E_{C,i}) \\ \Delta E_V &= \max(\Delta E_{V,i})\end{aligned}\quad (644)$$

The band gap and affinity can be modified:

$$\begin{aligned}E_g &= E_{g0} + \Delta E_C - \Delta E_V \\ \chi &= \chi_0 - \Delta E_C\end{aligned}\quad (645)$$

where the index '0' corresponds to the affinity and bandgap values before stress deformation.

Using Deformation Potential Model

To activate the deformation potential model, the name of the model must be specified in the Piezo section of the command file, for example:

```
Physics (Region = "StrainedSilicon") {
    Piezo(
        Model(DeformationPotential)
    )
}
```

To apply [Eq. 644](#) to the strain-induced conduction and valence band-edge shift, you must specify an additional option for the model: DeformationPotential(minimum). To activate $k \cdot p$ models ([Eq. 642](#) for electrons, or solving the cubic equation from [\[10\]](#) for holes), use DeformationPotential(ekp hkp).

All parameters of the model can be modified in the LatticeParameters section of the parameter file. The Sentaurus Device simulation coordinate system relative to the crystal orientation system can be defined by the X and Y vectors in this section. The default is X=(1 0 0), Y=(0 1 0). In addition, there is an option to represent the crystal system relative to the Sentaurus Device simulation system. In this case, the keyword CrystalAxis must be in the LatticeParameters section, and the X and Y vectors will represent the $\langle 100 \rangle$ and $\langle 010 \rangle$ axes of the crystal system in the Sentaurus Device simulation system.

The number of conduction and valence band subvalleys n_C and n_V are defined by NC and NV in the parameter file; by default, $n_C = 3$, $n_V = 2$. Appropriate deformation potential constants ξ_{ij}^C , ξ_{ij}^V , ξ_{ij}^{C2} , and ξ_{ij}^{V2} [eV] are defined in the fields DC[i], DV[i], DC2[i], and DV2[i] of the parameter file, respectively.

The elasticity modulus S_{ij} [10^{-12} cm²/dyn] can be specified in the field S[i][j] in the parameter file. If the cubic crystal system is selected (by specifying CrystalSystem=0), it is sufficient to specify S_{11} , S_{12} , and S_{44} . For a hexagonal crystal system (CrystalSystem=1), S_{33} and S_{13} must also be specified. Otherwise, all unspecified moduli are set to 0.

As an example, the following section represents silicon LatticeParameters defined for the Bir and Pikus model in the parameter file:

```
LatticeParameters{
    * Crystal system, elasticity, and deformation potential are defined.
    X = (1, 0, 0)          #[1]
    Y = (0, 1, 0)          #[1]
    S[1][1] = 0.77          # [1e-12 cm^2/dyn]
    S[1][2] = -0.21         # [1e-12 cm^2/dyn]
    S[4][4] = 1.25          # [1e-12 cm^2/dyn]
    CrystalSystem = 0        #[1]
    NC = 3                  #[1]
```

23: Modeling Mechanical Stress Effect

Deformation of Band Structure

```
NV = 2          # [1]
DC2(1) = 0.9,-8.6,-8.6,0,0,0
DC2(2) = -8.6,0.9,-8.6,0,0,0
DC2(3) = -8.6,-8.6,0.9,0,0,0
DV2(1) = -2.1,-2.1,-2.1,-1,0.5,4
DV2(2) = -2.1,-2.1,-2.1,1,0.5,4
}
```

To modify the deformation potentials of $k \cdot p$ models, the following parameters in the section `LatticeParameters` should be used:

```
* Deformation potentials of k.p model for electron bands
xis = 7      # [eV]
dbs = 0.53 # [eV]
xiu = 9.16 # [eV]
xid = 0.77 # [eV]
* Deformation potentials of k.p model for hole bands
adp = 2.1      # [eV]
bdp = -2.3300e+00 # [eV]
ddp = -4.7500e+00 # [eV]
dso = 0.044      # [eV]
```

The parameters `xis`, `dbs`, `xiu`, and `xid` correspond to the conduction band deformation potentials Ξ_u' , Δ , Ξ_u , Ξ_d in [Eq. 642, p. 597](#). The other parameters are the valence-band deformation potentials described in [\[10\]](#):

- `adp` is the hydrostatic deformation potential.
- `bdp` is the shear deformation potential.
- `ddp` is the deformation potential.
- `dso` is the spin-orbit splitting energy.

Strained Effective Masses and Density-of-States

Sentaurus Device provides options for computing strain-dependent effective mass for both electrons and holes and, consequently, the strain-dependent conduction band and valence band effective density-of-states (DOS).

Strained Electron Effective Mass and DOS

The conduction band in silicon is approximated by three pairs of equivalent Δ_2 valleys. Without stress, the DOS of each valley is:

$$N_{C,i} = \frac{N_C}{3}, \quad i = 1, 3 \quad (646)$$

where N_C can be defined by two effective mass components m_l and m_t (see [Eq. 152, p. 237](#)).

As described in [Deformation of Band Structure on page 596](#), an applied stress induces a relative shift of the energy $\Delta E_{C,i}$ that is different for each Δ_2 valley. In addition, in the presence of shear stress, there is a large effective mass change for electrons. An analytic derivation for this mass change can be found in [\[11\]](#) and is based on a two-band $k \cdot p$ theory. To simplify the final expressions, the same dimensionless off-diagonal strain introduced in [Eq. 642](#) is used here:

$$\eta_i = \frac{4\Xi_u \varepsilon'_{jk}}{\Delta}, \quad j \neq k \neq i \quad (647)$$

Note that the ε'_{jk} strain affects only the Δ_2 valley along the i -axis, where $i = 1, 2$, or 3 represents the [100], [010], or [001] axis, respectively.

When the $k \cdot p$ model [\[11\]](#) is evaluated at the band minima of the first conduction band, two different branches for the transverse effective mass are obtained where m_{t1}^i is the mass across the stress direction, and m_{t2}^i is the mass along the stress direction:

$$m_{t1}^i / m_t = \begin{cases} \left(1 - \frac{\eta_i}{M}\right)^{-1}, & |\eta_i| \leq 1 \\ \left(1 - \frac{\text{sign}(\eta_i)}{M}\right)^{-1}, & |\eta_i| > 1 \end{cases} \quad (648)$$

23: Modeling Mechanical Stress Effect

Strained Effective Masses and Density-of-States

$$m_{t2}^i/m_t = \begin{cases} \left(1 + \frac{\eta_i}{M}\right)^{-1}, & |\eta_i| \leq 1 \\ \left(1 + \frac{\text{sign}(\eta_i)}{M}\right)^{-1}, & |\eta_i| > 1 \end{cases} \quad (649)$$

$$m_l^i/m_l = \begin{cases} \left(1 - \eta_i^2\right)^{-1}, & |\eta_i| < 1 \\ \left(1 - \frac{1}{|\eta_i|}\right)^{-1}, & |\eta_i| > 1 \end{cases} \quad (650)$$

In the above equations, M is a $k \cdot p$ model parameter that has been adjusted to provide a good fit with the empirical pseudopotential method (EPM) results.

The stress-induced change of the effective DOS for each valley can then be written as:

$$N_{C,i} = \sqrt{\left(\frac{m_{t1}^i}{m_t}\right)\left(\frac{m_{t2}^i}{m_t}\right)\left(\frac{m_l^i}{m_l}\right)} \cdot \left(\frac{N_C}{3}\right) \quad (651)$$

Accounting for the change of the stress-induced valley energy $\Delta E_{C,i}$ and the carrier redistribution between valleys, the strain-dependent conduction-band effective DOS can be derived for Boltzmann statistics:

$$N'_C = \gamma \cdot N_C \quad (652)$$

where:

$$\gamma = \frac{1}{N_C} \cdot \sum_{i=1}^3 N_{C,i} \cdot \exp\left(\frac{\Delta E_C^{\min} - \Delta E_{C,i}}{kT_n}\right) \quad (653)$$

and:

$$\Delta E_C^{\min} = \min(\Delta E_{C,i}) \quad (654)$$

This is incorporated into Sentaurus Device as a strain-dependent electron effective mass using:

$$m'_n = \gamma^{2/3} \cdot m_n \quad (655)$$

and:

$$N'_C = \left(\frac{m'_n}{m_n}\right)^{3/2} N_C \quad (656)$$

Strained Hole Effective Mass and DOS

To compute the hole effective DOS mass for arbitrary strain in silicon, the band structure provided by the six-band $k \cdot p$ method is explicitly integrated assuming Boltzmann statistics [12][13]. In this approximation, the total hole effective DOS mass is given by:

$$m_p'(T) = [m_{cc,1}^{3/2} e^{-(E_3 - E_1)/(kT)} + m_{cc,2}^{3/2} e^{-(E_3 - E_2)/(kT)} + m_{cc,3}^{3/2}]^{2/3} \quad (657)$$

where E_1 , E_2 , and E_3 are the ordered band edges for each of the three valence valleys, and $m_{cc,1}$, $m_{cc,2}$, and $m_{cc,3}$ are the carrier-concentration masses for each of the valleys given by:

$$m_{cc}^{3/2}(T) = \frac{2}{\sqrt{\pi}} (kT)^{-3/2} \int_0^{\infty} dE m_{DOS}^{3/2}(E) \sqrt{E} e^{-E/(kT)} \quad (658)$$

The energy-dependent DOS mass of each valley, m_{DOS} , is given by:

$$m_{DOS}^{3/2}(E) = \frac{\sqrt{2}\pi^2 \hbar^3}{\sqrt{E}} \frac{1}{(2\pi)^3} \int_0^{2\pi} d\phi \int_0^{\pi} d\theta \sin(\theta) \left[\left| \frac{\partial k}{\partial E} \right| k^2 \right]_{\phi, \theta, E} \quad (659)$$

The band structure-related integrand is computed from the inverse six-band $k \cdot p$ method in polar k -space coordinates. The integrals in Eq. 658 and Eq. 659 are evaluated using optimized quadrature rules.

The six-band $k \cdot p$ method is controlled by seven parameters:

- Three Luttinger–Kohn parameters (γ_1 , γ_2 , γ_3) that determine the band dispersion.
- The spin-orbit split-off energy (Δ_{so}).
- Three deformation potentials (a, b, d) that determine the strain response.

The Luttinger–Kohn parameters and Δ_{so} have been set to reproduce the DOS mass for relaxed silicon, m_p , as given by Eq. 155, p. 239 as a function of temperature. The default deformation potentials have been taken from the literature [14].

The strain-dependent valence-band effective DOS is then calculated from:

$$N'_V = \left(\frac{m'_p}{m_p} \right)^{3/2} N_V \quad (660)$$

23: Modeling Mechanical Stress Effect

Strained Effective Masses and Density-of-States

Using Strained Effective Masses and DOS

The strain-dependent effective mass and DOS calculations can be selected by specifying DOS (eMass), DOS (hMass), or DOS (eMass hMass) as an argument to Piezo (Model ()) in the Physics section of the command file. For example:

```
Physics {
    Piezo (
        Model (
            DOS (eMass hMass)
        )
    )
}
```

Currently, these models have been calibrated only for strained silicon. Parameters affecting the DOS (eMass) model include the deformation potentials of the $k \cdot p$ model for electron bands (x_{is} , d_{bs} , x_{iu} , x_{id}) and the Sverdlov $k \cdot p$ parameter (M_{kp}). Parameters affecting the DOS (hMass) model include the deformation potentials of the $k \cdot p$ model for hole bands (a_{dp} , b_{dp} , d_{dp} , d_{so}) and the Luttinger parameters (γ_1 , γ_2 , γ_3). These parameters can be specified in the LatticeParameters section of the parameter file.

The DOS (hMass) model involves stress-dependent and lattice temperature-dependent numeric integrations that can be very CPU intensive. For simulations at a constant lattice temperature, you should only observe this CPU penalty once, usually at the beginning of the simulation. For nonisothermal thermal simulations, these integrations would usually have to be repeated for every lattice temperature change during the solution, resulting in a very prohibitive simulation.

As an alternative, Sentaurus Device provides an option for modeling the lattice temperature dependency of the strain-affected hole mass with analytic expressions that are fit to the full numeric integrations for each stress in the device. A CPU penalty will still be observed at the beginning of the simulation while fitting parameters are determined, but the remainder of the simulation should proceed as usual since the analytic expression evaluations are very fast.

By default, the analytic lattice temperature fit is used with thermal simulations and numeric integration is used for isothermal simulations. These defaults can be overridden by using the AnalyticLTFit or NumericalIntegration options for DOS (hMass):

```
DOS (eMass hMass (NumericalIntegration))
DOS (eMass hMass (AnalyticLTFit))
```

Multivalley Band Structure

For a general description of the multivalley model and its implementation, see [Multivalley Statistics on page 203](#).

For stress simulations, the multivalley model uses analytic bands computed by two-band $k \cdot p$ [11] and $6 \times 6 k \cdot p$ [10] models for electrons and holes, respectively. These models consider three separate valleys in both the conduction and valence bands. The valley energy shifts related to the models are described in [Deformation of Band Structure on page 596](#).

For electrons, the $k \cdot p$ model gives an analytic expression ([Eq. 642, p. 597](#) for $\Delta E_{C,i}$), which defines the valley energy shifts. However, for holes, it is based on a numeric solution of the $6 \times 6 k \cdot p$ cubic equation.

Computation of the effective DOS factors of [Eq. 97, p. 204](#) and [Eq. 98, p. 204](#) is based on the same $k \cdot p$ models and uses a stress-induced change of the effective DOS masses for each valley described in [Strained Effective Masses and Density-of-States on page 601](#). Referring to [Eq. 651](#) and [Eq. 657](#), and using their variables, the effective DOS factors of each valley can be expressed as follows:

$$g_n^i = \frac{1}{3} \sqrt{\left(\frac{m_{t1}^i}{m_t}\right) \left(\frac{m_{t2}^i}{m_t}\right) \left(\frac{m_l^i}{m_l}\right)} \quad g_p^i = \left(\frac{m_{cc,i}}{m_p}\right)^{3/2} \quad (661)$$

Using Multivalley Band Structure

For stress simulations (with the `Piezo` statement in the `Physics` section), the model ignores all valley specifications in the parameter file, which are described in [Using Multivalley Statistics on page 204](#). The model 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. The model allows you to simplify the `Piezo` section where the `DeformationPotential` and `DOS` statements should not be used.

NOTE Although the multivalley model can work together with the `DeformationPotential` model (it recomputes the valley energy shifts with reference to the band edges defined by the deformation potential model), it stops Sentaurus Device with an error message if the `DOS` statement is used (see [Strained Effective Masses and Density-of-States on page 601](#)).

23: Modeling Mechanical Stress Effect

Piezoresistance Mobility Model

NOTE Similar to DOS (hMass), the multivalley model defined for holes involves numeric integrations that can be very CPU intensive but, for the multivalley model, this integration is performed only once at the beginning of the simulation. This huge CPU penalty suggests excluding temperature dependency of the hole valley properties to have a practical simulation time with the lattice heat equation.

Piezoresistance Mobility Model

This approach [5][15] focuses on the modeling of the piezoresistive effect. The basic part of the model is a linear extension of the constitutive current relations for electrons and holes in single-crystalline silicon [15] for small stresses:

$$\begin{aligned}\bar{\mu}_\alpha &= \bar{\mu}_\alpha^0 (\bar{1} - \bar{\Pi}^\alpha \cdot \bar{\sigma}), \quad \alpha = n, p \\ \bar{j}_\alpha &= \bar{\mu}_\alpha \cdot \left(\frac{\bar{j}_\alpha^0}{\bar{\mu}_\alpha^0} \right)\end{aligned}\tag{662}$$

where:

- $\bar{\mu}_\alpha$ is the second rank mobility tensor.
- $\bar{\mu}_\alpha^0$ denotes the isotropic mobility without stress.
- $\bar{1}$ is the identity tensor.
- $\bar{\sigma}$ is the stress tensor.
- $\bar{\Pi}^\alpha$ is the tensor of piezoresistance coefficients that depend on the doping concentration and temperature distribution.
- \bar{j}_α^0 is the vector of the carrier current without the stress.

The stress tensor $\bar{\sigma}$ and mobility matrix $\bar{\mu}_\alpha$ are symmetric 3×3 matrices. Therefore, they have only six independent components.

With the index transformation rule (see [Overview on page 595](#)), they can be written in a six-component vector notation.

In crystals with cubic symmetry such as silicon, the number of independent coefficients of the piezoresistance tensor reduces to three by rotating the coordinate system parallel to the high-symmetric axes of the crystal [7]:

$$\bar{\Pi}^\alpha = \begin{bmatrix} \Pi_{11}^\alpha & \Pi_{12}^\alpha & \Pi_{12}^\alpha & 0 & 0 & 0 \\ \Pi_{12}^\alpha & \Pi_{11}^\alpha & \Pi_{12}^\alpha & 0 & 0 & 0 \\ \Pi_{12}^\alpha & \Pi_{12}^\alpha & \Pi_{11}^\alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & \Pi_{44}^\alpha & 0 & 0 \\ 0 & 0 & 0 & 0 & \Pi_{44}^\alpha & 0 \\ 0 & 0 & 0 & 0 & 0 & \Pi_{44}^\alpha \end{bmatrix} \quad (663)$$

Since the coordinate system of the simulation is not necessarily parallel to the high-symmetric axis of the crystal, the orientations of the x-axis and y-axis of the crystal system can be specified in the command file (see [Using Deformation Potential Model on page 599](#)).

External strain leads to a change in the effective masses and anisotropic scattering. The first effect is described by an independent constant term $\Pi_{ij, \text{con}}^\alpha$, but the second effect, the scattering, can be calculated [16] at room temperature for low-doping concentrations ($\Pi_{ij, \text{var}}^\alpha$) and can be multiplied by a doping-dependent and temperature-dependent factor $P_\alpha(N, T)$.

Both effects are considered in the piezoresistance coefficients by [16]:

$$\Pi_{ij}^\alpha = \Pi_{ij, \text{var}}^\alpha P_\alpha(N, T) + \Pi_{ij, \text{con}}^\alpha \quad (664)$$

In the case of electrons, the scalar mobility used in the drift-diffusion and hydrodynamic equations is a mean value averaged over the different conduction band minima. If the symmetry of the crystal is destroyed, for example by external strain, the conduction band valleys shift and, therefore, yield electron transfers between the valleys. This redistribution of electrons in the conduction band leads to anisotropic scattering.

In the case of holes, the mobility is an averaged quantity including heavy and light holes. External strain leads to a lift of the degeneracy at the valence band maximum. According to the literature [16], the doping-dependent and temperature-dependent factor $P_\alpha(N, T)$ can be given by:

$$P_\alpha(N, T) = \frac{300 \text{ K}}{T} \frac{F_0\left(\frac{E_{F,\alpha}}{kT}\right)}{F_0'\left(\frac{E_{F,\alpha}}{kT}\right)} \quad (665)$$

where $F_0(x)$ and $F'_0(x)$ are the Fermi integrals of the order 0 and its first derivative. The Fermi energy $E_{F,\alpha}$ is equal to $E_n - E_C$ for electrons and $E_V - E_p$ for holes. They are calculated

23: Modeling Mechanical Stress Effect

Piezoresistance Mobility Model

by using appropriate analytic approximations [17] where charge neutrality is assumed between carrier and doping (N), and it gives the doping dependence of the model. The numeric evaluation of $P_\alpha(N, T)$ is based on an analytic fit of the Fermi integrals [18].

The default values of the piezoresistance coefficients for low-doped silicon at 300 K are listed in [Table 97](#) and [Table 98](#). They can be changed in the parameter file of Sentaurus Device as described in [Using Piezoresistance Mobility Model](#).

Using Piezoresistance Mobility Model

The piezoresistance model is applied to a simulation by including the name of the model in the `Model` subsection of the `Piezo` section of the input command file. It can be specified as follows:

```
Physics { Piezo( Model(Mobility(Tensor)) ) }
```

The values of the piezoresistance coefficients can be changes in the Sentaurus Device parameter file with the following syntax:

```
Piezoresistance{
    p11var  $\Pi_{11}^n$ , var ,  $\Pi_{11}^p$ , var # [1/Pa]
    p12var  $\Pi_{12}^n$ , var ,  $\Pi_{12}^p$ , var # [1/Pa]
    p44var  $\Pi_{44}^n$ , var ,  $\Pi_{44}^p$ , var # [1/Pa]
    p11con  $\Pi_{11}^n$ , con ,  $\Pi_{11}^p$ , con # [1/Pa]
    p12con  $\Pi_{12}^n$ , con ,  $\Pi_{12}^p$ , con # [1/Pa]
    p44con  $\Pi_{44}^n$ , con ,  $\Pi_{44}^p$ , con # [1/Pa]
}
```

Table 97 Piezoresistive parameters for holes

Π_{11}^p , con	Π_{12}^p , con	Π_{44}^p , con	Π_{11}^p , var	Π_{12}^p , var	Π_{44}^p , var	Unit
5.1×10^{-11}	-2.6×10^{-11}	2.8×10^{-10}	1.5×10^{-11}	1.5×10^{-11}	1.1×10^{-9}	Pa^{-1}

Table 98 Piezoresistive parameters for electrons

Π_{11}^n , con	Π_{12}^n , con	Π_{44}^n , con	Π_{11}^n , var	Π_{12}^n , var	Π_{44}^n , var	Unit
0	0	0	-1.026×10^{-9}	5.34×10^{-10}	-1.36×10^{-10}	Pa^{-1}

The `Mobility` keyword can have different options: `Tensor`, `eTensor`, and `hTensor`, where ‘e’ or ‘h’ switches on the stress-dependent mobility model only for electrons or holes, respectively.

These options use only the constant piezoresistivity coefficients $\Pi_{ij}^\alpha = \Pi_{ij,\text{var}}^\alpha + \Pi_{ij,\text{con}}^\alpha$ where the doping-dependent and temperature-dependent factor [Eq. 665](#) is equal to 1. To activate the doping and temperature dependence [Eq. 664](#), add the keyword Kanda, for example, `Tensor(Kanda)`.

Enormal- and MoleFraction-dependent Piezo Coefficients

The measured data shows that the piezoresistive coefficients can have dependencies with respect to the normal electric field E_\perp or the mole fraction x or both. Sentaurus Device has a calibration option to specify the dependency of the piezoresistive coefficients.

This model is based on the piezoresistive prefactors $P_{ij}^\alpha = P_{ij}^\alpha(E_\perp, x)$. The new coefficients are equal to:

$$\Pi_{ij,\text{new}}^\alpha = P_{ij}^\alpha(E_\perp, x) \cdot \Pi_{ij}^\alpha \quad (666)$$

Sentaurus Device allows a piecewise linear or spline approximation (the third degree) of the prefactors over the normal electric field and a piecewise linear or piecewise cubic approximation over the mole fraction (see [Ternary Semiconductor Composition on page 552](#)).

Using Piezoresistive Prefactors Model

To activate this model, add the keyword `Enormal` to the subsection `{e,h}Tensor`, for example:

```
Physics {
    ...
    Piezo(
        Model(Mobility(eTensor(Kanda Enormal)))
    )
}
```

The implementation of this model is based on the PMI (see [Piezoresistive Coefficients on page 1026](#)). The keyword `Enormal` means that Sentaurus Device uses the PMI predefined model `PmiEnormalPiezoResist`. You can create your own PMI models and, in this case, the keyword `Enormal` must be replaced by the name of the PMI model (for example, `eTensor("my_pmi_model")`) and the parameter file must contain a corresponding section.

By default, the values of all prefactors are equal to 1. These values can be changed in the section `PmiEnormalPiezoResist` of the parameter file. The following examples show how to use this section.

23: Modeling Mechanical Stress Effect

Enormal- and MoleFraction-dependent Piezo Coefficients

Example 1

This example shows the section of the parameter file for purely Enormal-dependent prefactors:

```
Material = "Silicon" {
    * Example 1: Only Enormal dependence of piezoresistive coefficients
    PmiEnormPiezoResist
    {
        eEnormalFormula = 2 # cubic spline
        * = 0 # no Enormal dependence (default)
        * = 1 # piecewise linear approximation
        * = 2 # cubic spline approximation

        eNumberOfEnormalNodes = 3      # number of nodes for Spline(Enormal)

        # _k is _"index of node"
        eEnormal_1 = 1.0e+5 # [V/cm]
        eP11_1 = 1.0      # [1]
        eP12_1 = 1.0      # [1]
        eP44_1 = 1.0      # [1]

        eEnormal_2 = 4.0e+5 # [V/cm]
        eP11_2 = 0.75     # [1]
        eP12_2 = 0.75     # [1]
        eP44_2 = 0.75     # [1]

        eEnormal_3 = 7.0e+5 # [V/cm]
        eP11_3 = 0.5      # [1]
        eP12_3 = 0.5      # [1]
        eP44_3 = 0.5      # [1]

        hEnormalFormula = 1          # piecewise linear approximation
        hNumberOfEnormalNodes = 4 # number of nodes

        # _k is _"index of the node"
        hEnormal_1 = 1.0e+5 # [V/cm]
        hP11_1 = 1.0      # [1]
        hP12_1 = 1.0      # [1]
        hP44_1 = 1.0      # [1]

        hEnormal_2 = 1.9e+5 # [V/cm]
        hP11_2 = 0.969625 # [1]
        hP12_2 = 0.969625 # [1]
        hP44_2 = 0.969625 # [1]

        hEnormal_3 = 6.1e+5 # [V/cm]
        hP11_3 = 0.530375 # [1]
```

```

hP12_3 = 0.530375 # [1]
hP44_3 = 0.530375 # [1]

hEnormal_4 = 7.0e+5 # [V/cm]
hP11_4 = 0.5      # [1]
hP12_4 = 0.5      # [1]
hP44_4 = 0.5      # [1]
}
}
}

```

Figure 58 shows the dependency of these piezo prefactors with respect to Enormal.

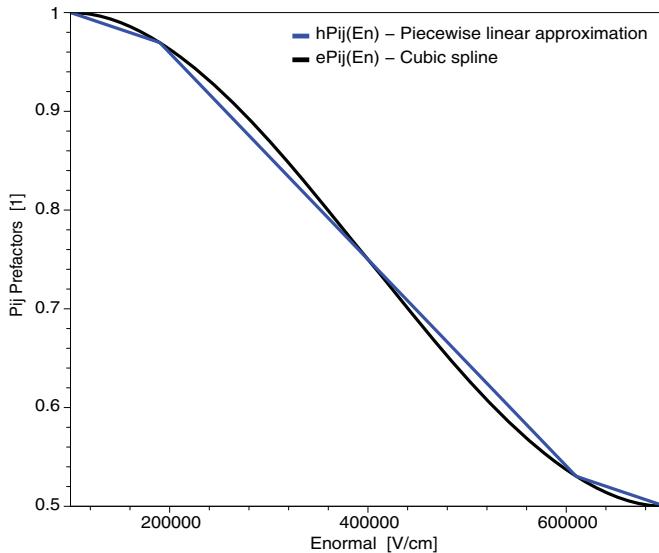


Figure 58 Example of cubic spline and piecewise linear approximation of piezo prefactors

Example 2

This example shows the section of the parameter file for purely MoleFraction-dependent prefactors:

```

Material = "SiliconGermanium" {
* Mole dependent material: SiliconGermanium (x=0) = Silicon
* Mole dependent material: SiliconGermanium (x=1) = Germanium
* Example 2: Only MoleFraction dependence.
PmiEnormPiezoResist
{
    eMoleFormula = 1           # piecewise linear approximation
    eMoleFractionIntervals = 1 # number of MoleFraction intervals
                                # i.e. x0=0, x1=1
                                # _i is _"index of mole fraction value"
}

```

23: Modeling Mechanical Stress Effect

Enormal- and MoleFraction-dependent Piezo Coefficients

```
eXmax_0 = 0
eP11_0 = 1 # [1]
eP12_0 = 1 # [1]
eP44_0 = 1 # [1]

eXmax_1 = 1
eP11_1 = 0.5 # [1]
eP12_1 = 0.5 # [1]
eP44_1 = 0.5 # [1]

hMoleFormula = 2      # piecewise cubic approximation
hMoleFractionIntervals = 2
    # see Ternary Semiconductor Composition on page 552
    # P = P[i-1] + A[i]*dx + B[i]*dx^2 + C[i]*dx^3
    # where:
    # A[i] = (P[i]-P[i-1])/dx[i] - B[i]*dx[i] - C[i]*dx[i]
    # dx[i] = xMax[i] - xMax[i-1]
    # dx     = x - xMax[i-1]
    # i = 1,...,nbMoleIntervals

    # _i is _"index of mole fraction value"
hXmax_0 = 0
hP11_0 = 1 # [1]
hP12_0 = 1 # [1]
hP44_0 = 1 # [1]

hXmax_1 = 0.5
hP11_1 = 2 # [1]
hP12_1 = 2 # [1]
hP44_1 = 2 # [1]

    # B and C coefficients
hB11_1 = 4.
hC11_1 = 0.
hB12_1 = 4.
hC12_1 = 0.
hB44_1 = 4.
hC44_1 = 0.

hXmax_2 = 1
hP11_2 = 3 # [1]
hP12_2 = 3 # [1]
hP44_2 = 3 # [1]

    # B and C coefficients
hB11_2 = -4.
hC11_2 = 0.
hB12_2 = -4.
```

```

    hC12_2 = 0.
    hB44_2 = -4.
    hC44_2 = 0.
}
}
}
```

Figure 59 shows the dependency of the hP_{ij} prefactors of the previous example with respect to MoleFraction.

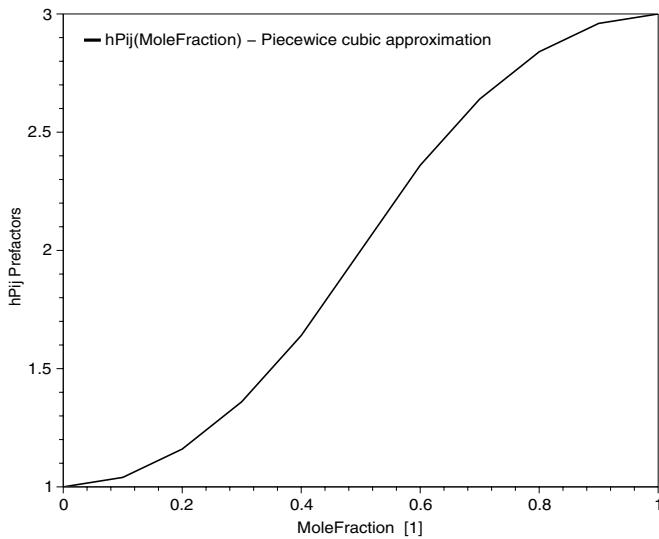


Figure 59 Example of piecewise cubic approximation

Example 3

This example shows the section of the parameter file for Enormal- and MoleFraction-dependent prefactors:

```

Material = "SiliconGermanium" {
  * Mole dependent material: SiliconGermanium (x=0) = Silicon
  * Mole dependent material: SiliconGermanium (x=1) = Germanium

  * Example 3: Enormal and MoleFraction dependent piezoresistance prefactors
  * (general case)
  Pm1EnormPiezoResist
  {

    eMoleFormula = 2           # piecewise cubic approximation
    eMoleFractionIntervals = 1 # i.e. x0=0, x1=1

    # begin description for mole fraction x0=0
    eXmax_0 = 0
    eEnormalFormula_0 = 2      # cubic spline
  }
}
```

23: Modeling Mechanical Stress Effect

Enormal- and MoleFraction-dependent Piezo Coefficients

```
eNumberOfEnormalNodes_0 = 3 # number of nodes for Spline(Enormal)

    # _i_k is _"index of mole fraction value"_ "index of node for Spline"
eEnormal_0_1 = 1.0e+5 # [V/cm]
    eP11_0_1 = 1.0 # [1]
    eP12_0_1 = 1.0 # [1]
    eP44_0_1 = 1.0 # [1]

eEnormal_0_2 = 4.0e+5 # [V/cm]
    eP11_0_2 = 0.75 # [1]
    eP12_0_2 = 0.75 # [1]
    eP44_0_2 = 0.75 # [1]

eEnormal_0_3 = 7.0e+5 # [V/cm]
    eP11_0_3 = 0.5 # [1]
    eP12_0_3 = 0.5 # [1]
    eP44_0_3 = 0.5 # [1]
# end description for mole fraction x0=0

# begin description for mole fraction x1=1
eXmax_1 = 1
eEnormalFormula_1 = 1      # piecewise linear approximation
eNumberOfEnormalNodes_1 = 2 # number of nodes

    # _i_k is _"index of mole fraction value"_ "index of node for Spline"
eEnormal_1_1 = 1.0e+5 # [V/cm]
    eP11_1_1 = 1.0 # [1]
    eP12_1_1 = 1.0 # [1]
    eP44_1_1 = 1.0 # [1]

eEnormal_1_2 = 7.0e+5 # [V/cm]
    eP11_1_2 = 0.5 # [1]
    eP12_1_2 = 0.5 # [1]
    eP44_1_2 = 0.5 # [1]

# B and C coefficients
hB11_1 = 3.
hC11_1 = -2.
hB12_1 = 3.
hC12_1 = -2.
hB44_1 = 3.
hC44_1 = -2.

# end description for mole fraction x1=1

# hPij prefactors are equal to default values (i.e. 1)
}
```

Figure 60 shows the difference between I_d - V_g curves for a MOSFET. The parameter file section `PmiEnormPiezoResist` is the same as in [Example 1 on page 610](#). The Physics section is:

```
Physics {
    Fermi
    Mobility( Phumob HighFieldsat Enormal )
    EffectiveIntrinsicDensity( OldSlotboom )
    Recombination( SRH(DopingDependence) )
    Piezo(
        Model( Mobility(eTensor(Kanda Enormal)) DeformationPotential )
        Stress = (1.3e8, 0, 0, 0, 0, 0)
    )
}
```

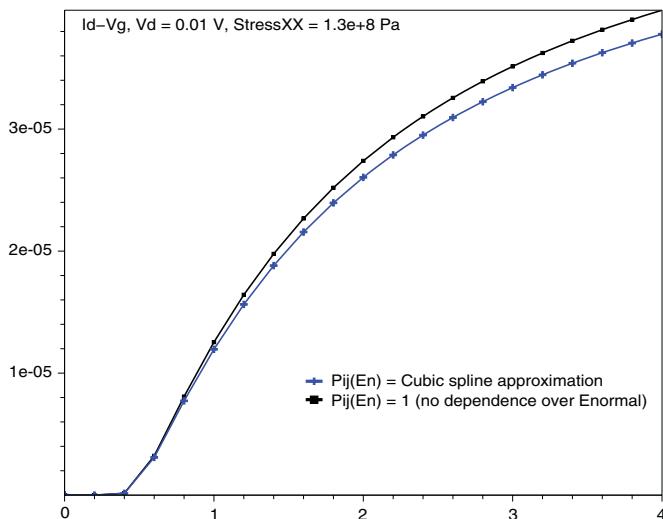


Figure 60 I_d - V_g curve, with Enormal-dependent piezo prefactors, shifts downwards

Piezoresistance Mobility Factor Models

Sentaurus Device provides capabilities for computing either first-order or second-order stress-dependent isotropic enhancement factors for low-field mobility. The first-order model accounts for a linear dependency on stress; whereas, the second-order model [19] accounts for both linear and quadratic dependencies on stress.

The factor models are based on the mobility enhancement tensor given by the expression:

$$\frac{\bar{\mu}_\alpha}{\mu_\alpha} = \bar{I} + \bar{\kappa}^{\alpha(1)} \cdot \bar{\sigma} + \bar{\kappa}^{\alpha(2)} \cdot (\bar{\sigma} \otimes \bar{\sigma}), \quad \alpha = n, p \quad (667)$$

23: Modeling Mechanical Stress Effect

Piezoresistance Mobility Factor Models

where $\bar{\kappa}^{\alpha(1)}$ and $\bar{\kappa}^{\alpha(2)}$ are first-order and second-order piezoconductance tensors, respectively, and \otimes is the tensor direct product. The components of the piezoconductance tensors are related to the components of the more familiar piezoresistance tensors through the following relations (given in Voigt notation, a standard method for expressing the components of symmetric tensors of even rank):

$$\begin{aligned}\Pi_{ij} &= -\kappa_{ij} & \leftrightarrow & \quad \kappa_{ij} = -\Pi_{ij} \\ \Pi_{ijk} &= -\kappa_{ijk} + \kappa_{ij}\kappa_{ik} & \leftrightarrow & \quad \kappa_{ijk} = -\Pi_{ijk} + \Pi_{ij}\Pi_{ik}\end{aligned}\tag{668}$$

Due to the crystal symmetry of silicon, there are only three independent first-order piezoresistance components (Π_{11} , Π_{12} , and Π_{44}) and only nine independent second-order piezoresistance components (Π_{111} , Π_{112} , Π_{122} , Π_{123} , Π_{144} , Π_{166} , Π_{661} , Π_{456} , and Π_{441}).

The model transforms the first-order and second-order piezoresistance tensors from the crystal system to the simulation system, and then computes piezoconductance tensor components using the relations in [Eq. 668](#).

To compute the mobility enhancement factor, it is assumed that the channel current is oriented in the x-direction of the device. With this assumption, the mobility enhancement factor can be computed from the piezoconductance tensor components in the simulation system using:

$$J_1^\alpha = J_{xx}^\alpha = 1 + \sum_j \kappa_{1j}^\alpha \sigma_j + \sum_{j,k} \kappa_{1jk}^\alpha \sigma_j \sigma_k\tag{669}$$

When the first-order model is used, only the first summation in [Eq. 669](#) is included in the mobility factor calculation.

As with the piezoresistance mobility tensor model (see [Piezoresistance Mobility Model on page 606](#)), the piezoresistance coefficients used in the first-order and second-order mobility factor calculations may include a doping-dependent and temperature-dependent factor $P_\alpha(N, T)$ (see [Eq. 665](#)).

Using Piezoresistance Mobility Factor Models

The first-order or second-order piezoresistance mobility factor model is applied to a simulation by including the keyword `Factor` in the `Mobility` statement of the `Piezo` model. The syntax for the piezoresistance mobility factor calculation, including options, is:

```
Physics {
    Piezo( Model ( Mobility ( Factor ( [FirstOrder | SecondOrder] Kanda ) ) )
}
```

The keyword `Factor` is a reminder that this model computes a mobility enhancement factor that is applied to the low-field mobility computed by Sentaurus Device (in contrast to an

enhancement tensor, which is obtained with the `Tensor` keyword as described in [Using Piezoresistance Mobility Model on page 608](#)). Specifying `Factor` computes enhancement factors for both electron and hole mobilities. To obtain an enhancement factor for only one carrier, the keyword `eFactor` or `hFactor` can be specified instead of `Factor`.

The specification of `FirstOrder` or `SecondOrder` is optional. The default is `SecondOrder`. If the `SecondOrder` model is used, Sentaurus Device uses the following second-order set of piezoresistance coefficients. These can be specified in the `Piezoresistance` section of the parameter file:

```
Piezoresistance {
    p11 = -1.1000e-09, 0.0000e+00 # [1/Pa]
    p12 = 4.5000e-10, 2.0000e-11 # [1/Pa]
    p44 = 2.5000e-10, 1.1900e-09 # [1/Pa]
    p111 = 6.6000e-19, -4.5000e-19 # [1/Pa^2]
    p112 = -5.5000e-20, 2.8000e-19 # [1/Pa^2]
    p122 = -2.2000e-20, -2.5000e-19 # [1/Pa^2]
    p123 = 8.8000e-19, 2.0000e-20 # [1/Pa^2]
    p144 = 1.0000e-20, -3.3000e-19 # [1/Pa^2]
    p166 = -6.9000e-19, 6.6000e-19 # [1/Pa^2]
    p661 = 6.0000e-21, -3.1000e-19 # [1/Pa^2]
    p456 = 2.0000e-20, -3.0000e-19 # [1/Pa^2]
    p441 = 2.0000e-20, 0.0000e+00 # [1/Pa^2]
}
```

If the `Kanda` option is specified with the `SecondOrder` model, the above piezoresistance coefficients are multiplied by the doping- and temperature-dependent factor $P_\alpha(N, T)$.

If the `FirstOrder` model is used, Sentaurus Device uses the same first-order set of piezoresistance coefficients that is used with the `Tensor` model, $\Pi_{ij, \text{var}}^\alpha$ and $\Pi_{ij, \text{con}}^\alpha$ (see [Using Piezoresistance Mobility Model on page 608](#)). If the `Kanda` option is also specified, the piezoresistance coefficients are modified according to [Eq. 664](#).

Device orientation with respect to the crystallographic system can be specified using the `X` and `Y` parameters in the `LatticeParameters` section of the parameter file. Stress tensors can be specified or read in from a file as described in [Using Stress-dependent Models on page 629](#).

Stress-induced Electron Mobility Model

Another approach [6] implemented in Sentaurus Device focuses on the modeling of the mobility changes due to the carrier redistribution between bands in silicon. As a known example, the electron mobility is enhanced in a strained-silicon layer grown on top of a thick, relaxed SiGe layer. Due to the lattice mismatch (which can be controlled by the Ge mole fraction), the thin silicon layer appears to be ‘stretched’ (under biaxial tension).

The origin of the electron mobility enhancement can be explained [6] by considering the six-fold degeneracy in the conduction band. The biaxial tensile strain lowers two perpendicular valleys (Δ_2) with respect to the four-fold in-plane valleys (Δ_4). Therefore, electrons are redistributed between valleys and Δ_2 is occupied more heavily. It is known that the perpendicular effective mass is much lower than the longitudinal one. Therefore, this carrier redistribution and reduced intervalley scattering enhance the electron mobility.

The model consistently accounts for a change of band energy as described in [Deformation of Band Structure on page 596](#), that is, a modification of the deformation potentials in [Eq. 641](#), [p. 597](#) will affect the strain-silicon mobility model. In the crystal coordinate system, the model gives only the diagonal elements of the electron mobility matrix.

The following expressions have been suggested for the electron mobility:

$$\mu_{ii}^n = \mu_n^0 \left[1 + \frac{1 - m_{nl}/m_{nt}}{1 + 2(m_{nl}/m_{nt})} \left(\frac{F_{1/2} \left(\frac{F_n - E_C - \Delta E_{C,i}}{kT} \right)}{F_{1/2} \left(\frac{F_n - E_C - \Delta E_C}{kT} \right)} - 1 \right) \right] \quad (670)$$

where:

- μ_n^0 is electron mobility without the strain.
- m_{nl} and m_{nt} are the electron longitudinal and transverse masses in the subvalley, respectively.
- $\Delta E_{C,i}$ and ΔE_C are computed by [Eq. 641](#) and [Eq. 643](#) or [Eq. 644](#), respectively.
- The index i corresponds to a direction (for example, μ_{11}^n is the electron mobility in the direction of the x-axis of the crystal system and, therefore, $\Delta E_{C,1}$ should correspond to the two-fold subvalley along the x-axis).
- F_n is quasi-Fermi level of electrons.

NOTE For the carrier quasi-Fermi levels, as for [Eq. 670](#), there are two options to be computed: (a) assuming the charge neutrality between carrier and doping, and it gives only the doping dependence of the model and (b) using the local carrier concentration (see [Using Stress-induced Electron Mobility Model on page 622](#)).

In the case of Boltzmann statistics, [Eq. 670](#) is simplified [6] and the dependence on the carrier (doping) concentration disappears:

$$\mu_{ii}^n = \mu_n^0 \left[1 + \frac{1 - m_{nl}/m_{nt}}{1 + 2(m_{nl}/m_{nt})} \left(\exp \left(\frac{\Delta E_C - \Delta E_{C,i}}{kT} \right) - 1 \right) \right] \quad (671)$$

Derivation of [Eq. 670](#) is based on consideration of the change in the stress-induced band energy in bulk silicon. However, in MOSFETs, there is an additional influence of a quantization that appears in the carrier channel at the silicon–oxide interface. For example, for electrons and

(100) surface orientation, there is a reduction of the stress effect with applied gate voltage. Partially, this is due to the Fermi-level dependency on the carrier concentration, but also the quantization in the channel may give a different carrier redistribution between electron bands. To account for this effect, Sentaurus Device provides an experimental option.

It is well known that, for an infinite triangular well with the electric field F , there is an analytic solution of the Schrödinger equation for eigenenergies and wave functions in the well. The eigenenergies represent a potential energy change of quantum carrier subbands and can be written as follows:

$$\varepsilon_{qm}^j \approx \left(\frac{\hbar^2}{2m}\right)^{1/3} \left[\frac{3}{2}\pi qF\left(j + \frac{3}{4}\right)\right]^{2/3} \quad (672)$$

[Eq. 672](#) suggests two things:

- The subband energy depends on the quantization mass as $m^{-1/3}$.
- It depends on the well electric field as $F^{2/3}$.

On the other hand, Sentaurus Device provides several options to account for quantization using the quantum correction models (see [Chapter 7 on page 251](#)) where the mass and field dependencies are accounted for assuming one ‘averaged’ band, as for example, in the density gradient model.

The quantum potential computed by these models (see [van Dort Quantization Model on page 252](#), [Density Gradient Quantization Model on page 260](#), [Modified Local-Density Approximation on page 264](#), and [1D Schrödinger Solver on page 253](#)) is a combination of all eigenenergies and wave functions that gives an ‘averaged’ band-edge shift due to the quantization.

With the experimental option, Sentaurus Device tries effectively to split the quantum potential (or to use [Eq. 672](#) with a local perpendicular electric field F_\perp) between the electrons bands to have the quantization accounted for in each band separately as follows:

$$\Delta E_{C,i}^{qm} = \alpha_i^\Lambda q \Lambda_n + \alpha_i^F |F_\perp|^{2/3} \quad (673)$$

where Λ_n is the quantum potential, and α_i^Λ , α_i^F are user-defined parameters described in [Using Stress-induced Electron Mobility Model on page 622](#). These parameters reflect an effective mass change between electron bands in a quantization direction. With this option, $\Delta E_{C,i} + \Delta E_{C,i}^{qm}$ is used in [Eq. 670](#) and [Eq. 671](#).

Intervalley Scattering

Another effect of the stress-induced mobility change is intervalley scattering. In the literature [20], the total relaxation time in valley i is expressed as follows:

$$\frac{1}{\tau_i} = \frac{1}{\tau_i^g} + \frac{1}{\tau_i^f} + \frac{1}{\tau_i^I} \quad (674)$$

where τ_i^g denotes the momentum relaxation time due to acoustic intravalley scattering and intervalley scattering between equivalent valleys (g-type scattering), τ_i^I is the impurity scattering relaxation time, and τ_i^f is the relaxation time for intervalley scattering between non-equivalent valleys (f-type scattering).

The intervalley scattering rate for electrons to scatter from initial valley i to final valley j can be defined as [20]:

$$\begin{aligned} S(\varepsilon, \Delta_{ij}) &= C \left[(\varepsilon - \Delta_{ij}^{\text{emi}})^{1/2} + \exp\left(\frac{\hbar\omega_{\text{opt}}}{kT}\right) (\varepsilon - \Delta_{ij}^{\text{abs}})^{1/2} \right] \\ \Delta_{ij}^{\text{emi}} &= \Delta E_{C,j} - \Delta E_{C,i} - \hbar\omega_{\text{opt}} \\ \Delta_{ij}^{\text{abs}} &= \Delta E_{C,j} - \Delta E_{C,i} + \hbar\omega_{\text{opt}} \end{aligned} \quad (675)$$

where $\hbar\omega_{\text{opt}}$ is the phonon energy and C is a constant. The relaxation time for this scattering event can be defined as follows:

$$\frac{1}{\tau_f(i \rightarrow j)} = \int_0^\infty S(\varepsilon, \Delta_{ij}) f(\varepsilon, F_n) d\varepsilon \quad (676)$$

where $f(\varepsilon, F_n)$ is the electron distribution function.

Using the Fermi–Dirac distribution function and considering that electrons from the valley i can be scattered into two valleys j and l ($1/\tau_i^f = 1/\tau_j^f(i \rightarrow j) + 1/\tau_l^f(i \rightarrow l)$), a ratio between unstrained and strained relaxation times for this intervalley scattering in valley i can be written as:

$$h_i = \frac{\tau_i^{f0}}{\tau_i^f} = \frac{F_{1/2}\left(\frac{\eta - \Delta_{ij}^{\text{emi}}}{kT}\right) + F_{1/2}\left(\frac{\eta - \Delta_{il}^{\text{emi}}}{kT}\right) + \exp\left(\frac{\hbar\omega_{\text{opt}}}{kT}\right) \left[F_{1/2}\left(\frac{\eta - \Delta_{ij}^{\text{abs}}}{kT}\right) + F_{1/2}\left(\frac{\eta - \Delta_{il}^{\text{abs}}}{kT}\right) \right]}{2 \left[F_{1/2}\left(\frac{\eta + \hbar\omega_{\text{opt}}}{kT}\right) + \exp\left(\frac{\hbar\omega_{\text{opt}}}{kT}\right) F_{1/2}\left(\frac{\eta - \hbar\omega_{\text{opt}}}{kT}\right) \right]} \quad (677)$$

where $\eta = F_n - E_C$.

NOTE The authors in [20] used the Boltzmann distribution function to express h_i . As a result, Eq. 20 of [20] does not have any doping dependence, but Eq. 677 does have it through the doping dependence of the Fermi level F_n .

Considering undoped/doped and unstrained/strained cases, the paper [20] derives an expression for total mobility change in valley i , which is based on a doping-dependent mobility model. With simplified doping dependence, a ratio between strained and unstrained total relaxation times for the valley i can be expressed as follows:

$$\frac{\tau_i}{\tau^0} = \frac{1}{1 + (h_i - 1) \frac{1 - \beta^{-1}}{1 + \left(\frac{N_{\text{tot}}}{N_{\text{ref}}}\right)^\alpha}} \quad (678)$$

where N_{tot} is the sum of donor and acceptor impurities, and $\beta = 1 + \tau^g / \tau^0$ is a fitting parameter [20] as both others N_{ref} and α .

The final modification of the mobility along valley i , which includes the stress-induced carrier redistribution and change in the intervalley scattering, is:

$$\mu_{ii}^n = \frac{3\mu_n^0}{1 + 2(m_{nl}/m_{nt})} \cdot \frac{\frac{\tau_i}{\tau^0} F_{1/2}\left(\frac{\eta - \Delta E_{C,i}}{kT}\right) + \frac{\tau_j m_{nl}}{\tau^0 m_{nt}} F_{1/2}\left(\frac{\eta - \Delta E_{C,j}}{kT}\right) + \frac{\tau_l m_{nl}}{\tau^0 m_{nt}} F_{1/2}\left(\frac{\eta - \Delta E_{C,l}}{kT}\right)}{F_{1/2}\left(\frac{\eta - \Delta E_{C,i}}{kT}\right) + F_{1/2}\left(\frac{\eta - \Delta E_{C,j}}{kT}\right) + F_{1/2}\left(\frac{\eta - \Delta E_{C,l}}{kT}\right)} \quad (679)$$

Effective Mass

It is known that the effective mass of electrons is not changed significantly if the stress is applied along the crystal axis. However, an analysis based on the empirical nonlocal pseudopotential theory [21] shows that, for the stresses applied along $\langle 110 \rangle$, the effective mass is changed strongly and it affects the electron mobility. The paper [21] suggests a simple empirical polynomial approximation for the dependence of the effective mass on stress applied along $\langle 110 \rangle$.

23: Modeling Mechanical Stress Effect

Stress-induced Electron Mobility Model

To account for this effect in device simulations, a logical way is to modify masses in [Eq. 679](#). In the literature [21], the stress-induced change of an average effective mass with parallel and perpendicular components to the stress along $\langle 110 \rangle$ has been formulated. The electron mobility change due to this effect can be expressed as follows:

$$\begin{aligned}\Delta\mu_{110}^n &= \mu_n^0 \left(\frac{m_{n10}}{m_{n10} + s_{110}\sigma_{110}} - 1 \right) \\ \Delta\mu_{-110}^n &= \mu_n^0 \left(\frac{m_{n10}}{m_{n10} + s_{-110}\sigma_{110}} - 1 \right) \\ \Delta\mu_{001}^n &= \mu_n^0 \left(\frac{m_{nt0}}{m_{nt0} + s_{001}(\sigma_{110})^2} - 1 \right)\end{aligned}\quad (680)$$

where σ_{110} is the stress along $\langle 110 \rangle$, $s_{110}, s_{-110}, s_{001}$ are user-defined parameters, and m_{n10}, m_{nt0} are unstressed effective masses that should be equal to m_n, m_{nt} of [Eq. 679](#). However, for flexibility, you can modify it separately in the Sentaurus Device parameter file.

NOTE Sentaurus Device uses a value of shear component of stress in the crystal system and [Eq. 680](#) is applied to each of the three planes with a rotation of the mobility tensor from [Eq. 680](#) to the crystal system. For example, in the xy plane σ_{110} is equal to the xy shear component of the stress multiplied by 2 ($2\sigma_{xy}$). Such representation is not general enough and the model should be used only for cases with a dominated $\langle 110 \rangle$ component of the stress.

Using Stress-induced Electron Mobility Model

The stress-induced mobility model can be activated regionwise or materialwise with the following keyword in the Mobility statement of the Piezo model:

```
Physics {  
    Piezo( Model(Mobility(eSubband)) )  
}
```

The keyword Subband assumes Boltzmann statistics and, in this case, [Eq. 671](#) is used. To activate doping dependency and [Eq. 670](#), the keywords eSubband(Doping) should be used, but if you need to influence the Fermi level on carrier redistribution between bands (for example, in the MOSFET channel), then use eSubband(Fermi).

The model parameters (see [Eq. 670](#)) can be specified in the parameter file in the `StressMobility` section as follows:

```
StressMobility {
    me_lt = 4.81      # [1]
    elec_100 = 1      # [1]
    elec_010 = 2      # [1]
    elec_001 = 3      # [1]
    mh_lh = 0.32653   # [1]
}
```

where `me_lt` is the ratio m_{nl}/m_{nt} . The numbers `elec_100`, `elec_010`, and `elec_001` correspond to indexes of deformation potentials in the `LatticeParameters` section (see [Using Deformation Potential Model on page 599](#)) for two-fold electron subvalleys in the direction $[100]$, $[010]$, and $[001]$, respectively.

To activate the experimental option to account for the quantization effect ([Eq. 673](#)) in the MOSFET channel, use the keyword `eSubband(Enormal)`. The parameters of the model can be specified in the parameter file with the following syntax:

```
StressMobility {
    dQC      = (0.2, -1.0000e-01, 0.2) #[1]
    dQC_neg = 0.0000e+00 # [1]
    dEnorm  = (0.0000e+00, 0.0000e+00, 0.0000e+00) #[eV/(V/cm)^1/3]
}
```

where:

- `dQC` corresponds to α_i^A .
- `dEnorm` corresponds to α_i^F .
- `dQC_neg` is a multiplication factor for α_i^A if the quantum potential is negative (the negative quantum potential reflects a penetration of the wave function through the barrier).

NOTE The parameters α_i^A and α_i^F are related to a quantization direction and surface orientation. The default above corresponds to (100) surface orientation and the 2D simulation case. For example, to have the same quantization effect in 3D case, the specification of `dQC` would be:
`dQC = (0.2, 0.2, -0.1).`

To activate the intervalley scattering model with Boltzmann statistics (no doping and carrier concentration dependency), use the keywords `eSubBand(Scattering)`. However, to activate doping dependency in both scattering and carrier redistribution ([Eq. 679](#)), use `eSubBand(Doping Scattering)`.

23: Modeling Mechanical Stress Effect

Intel Stress-induced Hole Mobility Model

The parameters of the intervalley scattering model can be specified in the same `StressMobility` section of the parameter file:

```
StressMobility {  
    Ephonon = 0.06      # [eV]  
    beta = 1.22        # [1]  
    Nref = 2.0000e+17  # [cm^-3]  
    alpha = 0.65       # [1]  
}
```

where `Ephonon` is $\hbar\omega_{\text{opt}}$ in [Eq. 675](#), and `beta`, `Nref`, and `alpha` are parameters of [Eq. 678](#).

To activate the model for stress-induced change of the electron effective mass, the following keywords should be used: `Mobility(eSubband(EffectiveMass))`. The model works with both the `eTensor` and `eSubBand` models.

Parameters of the model (see [Eq. 680](#)) can be specified in the parameter file as follows:

```
StressMobility {  
    me_l0 = 0.914      # [1]  
    me_t0 = 0.196      # [1]  
    st110 = -1.6000e-11 # [1/Pa]  
    st_110 = 2.9000e-11 # [1/Pa]  
    st001 = 2.3600e-20  # [1/Pa^2]  
}
```

Intel Stress-induced Hole Mobility Model

Intel [\[22\]](#) suggested a mobility model for strained PMOS devices based on the occupancy of different parts of the topmost valence band. As shown in [Figure 61 on page 625](#), under zero stress, the topmost valence band has a fourfold symmetry with arms along $\langle 110 \rangle$ and $\langle \bar{1}10 \rangle$. Each ellipsoid is characterized by a transverse mass m_t and a longitudinal mass m_l .

As compressive uniaxial stress along $\langle 110 \rangle$ is applied, one of the arms shrinks and the band becomes a single ellipsoidal band. In [Figure 61](#), this modification of the band structure is modeled as the splitting and change in curvature of two ellipsoidal bands. With applied stress, the maximum energy associated with each of these bands splits with carriers preferentially occupying the upper valley. In addition, the transverse and longitudinal effective masses of each of these two valleys change with stress.

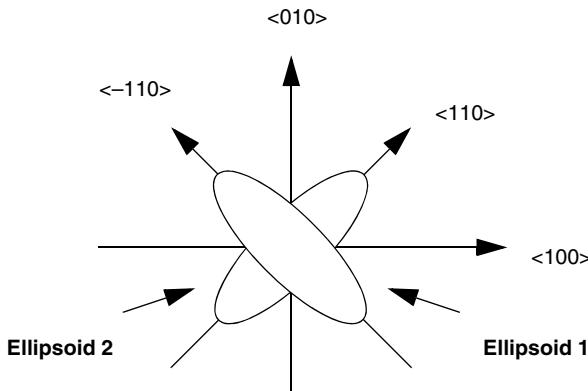


Figure 61 Sketch of Intel two-ellipsoid model; one ellipsoid is aligned along $\langle 110 \rangle$ and the other along $\langle -110 \rangle$

Under zero stress, the occupancies of the two ellipsoids are assumed equal and the mobility is isotropic with a value:

$$\mu_0 = q \langle \tau \rangle \left[\frac{0.5}{m_{t0}} + \frac{0.5}{m_{l0}} \right] \quad (681)$$

where $\langle \tau \rangle$ is the scattering time and the index ‘0’ in effective masses corresponds to the unstressed case.

Accounting for stress-induced reoccupation of these ellipsoids f_1 and f_2 , and assuming that the scattering time $\langle \tau \rangle$ is the same for both valleys and is independent of stress, the relative change in mobility is given by a diagonal tensor in the $\langle 110 \rangle$, $\langle \bar{1}10 \rangle$ coordinate system as [22]:

$$\begin{bmatrix} 1 + \frac{\Delta\mu_{110}}{\mu_0} \\ 1 + \frac{\Delta\mu_{-110}}{\mu_0} \end{bmatrix} = \frac{2m_{l0}m_{t0}}{m_{l0} + m_{t0}} \begin{bmatrix} \frac{f_1}{m_{t1}} + \frac{f_2}{m_{l2}} & 0 \\ 0 & \frac{f_1}{m_{l1}} + \frac{f_2}{m_{t2}} \end{bmatrix} \quad (682)$$

Denoting the energy split between the two values as Δ , the occupancies for the two valleys are given by:

$$\begin{aligned} f_1 &= \frac{1}{1 + \exp(-\Delta/kT)} \\ f_2 &= 1 - f_1 \end{aligned} \quad (683)$$

where T is the lattice temperature.

Stress Dependencies

Intel decomposes the planar stress tensor in the crystallographic coordinate system as:

$$S = \begin{bmatrix} b+a & s \\ s & b-a \end{bmatrix} \quad (684)$$

where b is the biaxial component, a is the anti-symmetric component, and s is the shear component. The basic model parameters ($1/m_t$ and Δ) are expanded in powers of these components. The longitudinal mass m_l is assumed to be stress independent, that is, $m_{l1} = m_{l2} = m_{l0}$.

The symmetry of the top valence band enforces some consistency relations on the basic parameters and its power-law expansions. For example, the mobility enhancement along $\langle 110 \rangle$, when uniaxial stress along $\langle 110 \rangle$ is applied, must be equal to the enhancement along $\langle \bar{1}10 \rangle$ when uniaxial stress along $\langle \bar{1}10 \rangle$ is applied. In addition, when biaxial stress is applied, the mobility enhancements along $\langle 110 \rangle$ and $\langle \bar{1}10 \rangle$ must be the same.

Enforcing these symmetries, the following expressions can be written:

$$\begin{aligned} \Delta &= d_1 s \\ \frac{1}{m_{t1}} &= \frac{1}{m_{t0}}(1 + s_{t1}s + s_{t2}s^2 + b_{t1}b + b_{t2}b^2) \\ \frac{1}{m_{t2}} &= \frac{1}{m_{t0}}(1 - s_{t1}s + s_{t2}s^2 + b_{t1}b + b_{t2}b^2) \end{aligned} \quad (685)$$

where the fitting parameters $d_1, s_{t1}, s_{t2}, b_{t1}, b_{t2}$ can be specified in the `StressMobility` section of the parameter file. The default values of these parameters are based on fitting the model to various PMOSFET stress data described in the literature [23].

Generalization of Model

The original Intel model considered only 2D planes. Therefore, it was extended and generalized in several issues: the three-dimensional case, doping dependence, and carrier redistribution between more than two valleys.

To generalize the model for the three-dimensional case, three $\langle 100 \rangle$ planes where the model is applied separately are considered. It is assumed that the valence band is a sum of six ellipsoids and this is consistent with the model suggested in the literature [24]. Sentaurus Device transforms the stress tensor into the crystal system and then, considering these three $\{100\}$ planes, it selects only the corresponding components of the stress tensor. For example, for the $[100]$, $[010]$ plane, Sentaurus Device takes only the s_1 , s_2 , and s_6 stress components

and recomputes them into a , b , and s of Eq. 684. Additionally, a modification of the effective mass perpendicular to the plane (parallel to the [001] direction) was introduced to account better for $\langle 001 \rangle$ simulation cases: $1/m_{t\langle 001 \rangle} = (1 + b_{tt}b)/m_{t0}$, where the parameter b_{tt} can be modified in the parameter file.

With this modification, the diagonal tensor of the stress-induced mobility change in the $\langle 110 \rangle$, $\langle \bar{1}10 \rangle$, $\langle 001 \rangle$ coordinate system can be written as:

$$\begin{bmatrix} \Delta\mu_{110} \\ \Delta\mu_{-110} \\ \Delta\mu_{001} \end{bmatrix} = \mu_0 \begin{bmatrix} \left(\frac{f_1}{m_{t1}} + \frac{f_2}{m_{t2}}\right)/\left(\frac{0.5}{m_{t0}} + \frac{0.5}{m_{t0}}\right) - 1 & 0 & 0 \\ 0 & \left(\frac{f_1}{m_{t1}} + \frac{f_2}{m_{t2}}\right)/\left(\frac{0.5}{m_{t0}} + \frac{0.5}{m_{t0}}\right) - 1 & 0 \\ 0 & 0 & m_{t0}/m_{t\langle 001 \rangle} - 1 \end{bmatrix} \quad (686)$$

Next, the relative change of the mobility is computed independently for each plane and it is summed in the crystal system. Finally, the mobility change tensor is transformed from the crystal system to the simulation one.

The carrier occupation of the two valleys (see Eq. 683) is derived assuming Boltzmann statistics and the same density-of-states in both these valleys. To obtain a doping dependence, it is necessary to consider Fermi–Dirac statistics and, in this case, the following expression is obtained:

$$f_1 = \frac{1}{1 + F_{1/2}\left(\frac{E_V - F_p - \Delta}{kT}\right)/F_{1/2}\left(\frac{E_V - F_p}{kT}\right)} \quad (687)$$

$$f_2 = 1 - f_1$$

where F_p is the hole quasi-Fermi level that is computed either assuming charge neutrality between carrier and doping, which gives only the doping dependency of the model or using carrier local concentration.

As previously discussed, the generalized model considers six ellipsoids and, therefore, the carrier reoccupation between all these valleys must be accounted for. This is not a simple problem because the stress-induced energy shift of each valley must be accounted for.

23: Modeling Mechanical Stress Effect

Intel Stress-induced Hole Mobility Model

As an experimental option, Sentaurus Device gives the following simplified expressions:

$$f_1 = \frac{F_{1/2}\left(\frac{E_V - F_p}{kT}\right)}{(N_e - 1)F_{1/2}\left(\frac{E_V - F_p}{kT}\right) + F_{1/2}\left(\frac{E_V - F_p - \Delta}{kT}\right)}$$
$$f_2 = \frac{F_{1/2}\left(\frac{E_V - F_p - \Delta}{kT}\right)}{(N_e - 1)F_{1/2}\left(\frac{E_V - F_p}{kT}\right) + F_{1/2}\left(\frac{E_V - F_p - \Delta}{kT}\right)}$$
 (688)

where N_e is equal to the number of ellipsoids that you want to consider. The value of N_e can be specified in the parameter file. The default value is 2, which transforms Eq. 688 into Eq. 687.

NOTE For this multi-ellipsoid option and $N_e > 2$, the sum $f_1 + f_2 < 1$ even without the stress. Therefore, Eq. 686 is slightly modified also to account for this different initial occupation.

Using Intel Mobility Model

To select the Intel stress-induced hole mobility model, you must include the following keyword in the Mobility statement of the Piezo model:

```
Physics {
    Piezo( Model(Mobility(hSixBand)) )
}
```

The keyword hSixBand assumes Boltzmann statistics and, in this case, Eq. 683 will be activated. To have doping dependence (see Eq. 687), the keyword hSixBand(Doping) should be specified, but to have carrier concentration dependency (Fermi statistics), use the keyword hSixBand(Fermi). The model parameters (see Eq. 685) can be specified in the StressMobility section of the parameter file as follows:

```
StressMobility {
    mh_10 = 0.48      # [1]
    mh_t0 = 0.15      # [1]
    ne = 2            # [1]
    d1 = -6.0000e-11  # [eV/Pa]
    st1 = -9.4426e-10 # [1/Pa]
    st2 = 4.3066e-19  # [1/Pa^2]
    bt1 = -1.0086e-10 # [1/Pa]
    bt2 = 6.5886e-21  # [1/Pa^2]
    btt = 1.2000e-10  # [1/Pa]
}
```

PMOS transistors are usually oriented in the direction to have maximum stress effect. To define this direction for the x-axis of a Sentaurus Device simulation, the following parameter set can be specified for the 2D case:

```
LatticeParameters {
    X = (1, 0, 1) #[1]
    Y = (0, 1, 0) #[1]
}
```

The hSixBand carrier occupancies f_1 and f_2 are calculated in the crystallographic coordinate system for each band. To plot these values, use the following keywords in the Plot section of the command file:

- f1BandOccupancy001 = Occupancy of the $\langle \bar{1}10 \rangle$ ellipsoid in the (001) plane
- f2BandOccupancy001 = Occupancy of the $\langle 110 \rangle$ ellipsoid in the (001) plane
- f1BandOccupancy010 = Occupancy of the $\langle \bar{1}10 \rangle$ ellipsoid in the (010) plane
- f2BandOccupancy010 = Occupancy of the $\langle 110 \rangle$ ellipsoid in the (010) plane
- f1BandOccupancy100 = Occupancy of the $\langle \bar{1}10 \rangle$ ellipsoid in the (100) plane
- f2BandOccupancy100 = Occupancy of the $\langle 110 \rangle$ ellipsoid in the (100) plane

Using Stress-dependent Models

To activate any of the stress-dependent models described in this section, the keyword Piezo must be specified in the Physics section of the input command file. Optionally, it can contain stress component specifications (if stress is constant over the device or region), and the components $\vec{\epsilon}_1$ OriKddX and $\vec{\epsilon}_2$ OriKddY of the coordinate system where the stress is defined (see [Table 99 on page 630](#)):

```
Physics {Piezo(
    Stress = (XX, YY, ZZ, YZ, XZ, XY)
    OriKddX = (1,0,0)
    OriKddY = (0,1,0)
)
}
```

NOTE The stress system is always defined relative to the simulation coordinate system of Sentaurus Device (in the Piezo section of the input file). The simulation coordinate system is defined relative to the crystal orientation system by default but, in the parameter file (see [Using Deformation Potential Model on page 599](#)), it is possible to define the crystal system relative to the simulation system. By default, all three coordinate systems coincide.

23: Modeling Mechanical Stress Effect

Using Stress-dependent Models

Table 99 General keywords for Piezo

Parameter	Description
Stress=(XX, YY, ZZ, YZ, XZ, XY)	Specifies uniform stress [Pa] if the Piezo file is not given in the File section.
OriKddX = (1, 0, 0)	Defines Miller indices of the stress system relative to the simulation system.
OriKddY = (0, 1, 0)	Defines Miller indices of the stress system relative to the simulation system.
Model(<options>)	Selects stress-dependent models in <options> (see Deformation of Band Structure on page 596 to Intel Stress-induced Hole Mobility Model on page 624).

There are two ways to define position-dependent stress values. In one option, a field of stress values [Pa] (as obtained by mechanical structure analysis) is read by specifying the Piezo entry in the File section:

```
File { ...
    Piezo = <piezofile>
}
```

Otherwise, a physical model interface can be used for stress specification. Optional parameters, which depend on the selected stress model, are described in the following sections.

NOTE Stress values in all these stress specifications should be in Pa (1 Pa = 10 dyn/cm²) and tensile stress should be positive according to convention.

Tensor Grid Option

Due to an applied mechanical stress, the mobility can become a tensor. The numeric approximation of the transport equations with tensor mobility is complicated (see [Chapter 21 on page 575](#)). However, if the mesh is a tensor one, the approximation is simpler. For this option, off-diagonal mobility elements are not used and, therefore, there are no mixed derivatives in the approximation. Such an approximation gives an M-matrix property for the Jacobian and it permits stable stress simulations. Very often, critical regions are simple and the mesh constructed in such regions can be close to a tensor one.

NOTE The off-diagonal elements of the mobility tensor appear only if there is a shear stress or the simulation coordinate system of Sentaurus Device is different from the crystal system.

To activate this simple tensor-grid approximation, the keyword `TensorGridAniso` in the `Math` section must be specified. By default, Sentaurus Device uses the `AverageAniso` approximation (see [Chapter 21 on page 575](#)) which is based on a local transformation of an anisotropic task (stress-induced mobility tensor) to an isotropic one.

NOTE Both approximation options do not guarantee a correct solution for arbitrary mesh and stress. Experiments with both these options can give an estimation of ignored terms.

Mobility Enhancement Limits

With high stress values, the stress-dependent mobility models in Sentaurus Device can sometimes cause the mobility to become negative or, in some cases, to become unrealistically large. This is particularly true for the piezoresistance mobility models. To prevent this, minimum and maximum stress enhancement factors for both electron and hole mobility can be specified in the `Piezoresistance` section of the parameter file. The following example shows the default values for `MinStressFactor` and `MaxStressFactor`:

```
Piezoresistance {
    MinStressFactor = 1e-5 , 1e-5 # [1]
    MaxStressFactor = 10   , 10   # [1]
}
```

NOTE Although these parameters are specified in the `Piezoresistance` section of the parameter file, these limits are applied to all stress-dependent mobility models.

Plotting Mobility Enhancement Factors

The mobility multiplication tensor ($\bar{I} - \Pi \cdot \bar{\sigma}$ in [Eq. 662](#) or tensors in [Eq. 670](#) and [Eq. 686](#)) can be plotted on the mesh nodes. This tensor is a symmetric 3×3 matrix and, therefore, has six independent values. To plot these values on the mesh nodes for electron and hole mobilities, use the keywords:

- `eMobilityStressFactorXX`, `eMobilityStressFactorYY`,
`eMobilityStressFactorZZ`
- `eMobilityStressFactorYZ`, `eMobilityStressFactorXZ`,
`eMobilityStressFactorXY`
- `hMobilityStressFactorXX`, `hMobilityStressFactorYY`,
`hMobilityStressFactorZZ`
- `hMobilityStressFactorYZ`, `hMobilityStressFactorXZ`,
`hMobilityStressFactorXY`

Stress Mobility Model for Minority Carriers

The measured data shows that stress dependence of minority carrier mobility (like carriers in the channel of MOSFETs) is different. For example, the stress effect for such carriers may have electric field and different (or smaller) doping dependencies. To have the possibility of a calibration, Sentaurus Device provides an option for an additional factor β for the stress effect applied to minority carrier mobility:

$$\bar{\mu}_\alpha = \mu_\alpha^0 \left(1 + \beta \frac{\Delta \mu_\alpha^{\text{Stress}}}{\mu_\alpha^0} \right), \quad \alpha = n, p \quad (689)$$

where $\Delta \mu_\alpha^{\text{Stress}}$ is the stress-induced tensor of the mobility change for any stress model described in this chapter. The parameter β can be specified (the default value is equal to 1) in the `Piezo` section of the input command file, for example:

```
Physics {
    Piezo( Model(Mobility(eMinorityFactor = 0.5 hMinorityFactor = 0.5) )
}
```

In some cases, it is useful to switch off doping dependence of stress models and to have this factor working partially for majority carriers (for example, in low-doped parts of MOSFET source/drain regions). This can be achieved by using the `ThresholdDoping` parameter as follows:

```
Physics {
    Piezo( Model(Mobility(eMinorityFactor(ThresholdDoping=1e18) = 0.5
                                hMinorityFactor(ThresholdDoping=-1e18) = 0.5) )
}
```

NOTE Switching the doping dependency in stress models means that `Tensor(Kanda)` will work as `Tensor`, `eSubBand(Doping)` will work as `eSubBand`, and `hSixBand(Doping)` will work as `hSixBand`.

NOTE `ThresholdDoping=0` corresponds to the regular definition of minority and majority carriers. If the factor β is applied to a part of the majority carriers, `ThresholdDoping` should be more than zero for electrons and less than zero for holes. A specification of `eMinorityFactor=0.5` is not equal to the specification `eMinorityFactor(ThresholdDoping=0)=0.5` because, in the latter case, the stress-doping dependency is switched off for carriers where this factor is applied.

Dependency of Saturation Velocity on Stress

Eq. 689 can be rewritten in the following form (see [Current Densities on page 577](#)):

$$\bar{\mu} = \mu^0 Q \begin{bmatrix} 1 + t_1 & 0 & 0 \\ 0 & 1 + t_2 & 0 \\ 0 & 0 & 1 + t_3 \end{bmatrix} Q^T \quad (690)$$

where:

- t_i are eigenvalues of the tensor $\beta \frac{\Delta \mu}{\mu^0}$.
- Q is the orthogonal matrix from eigenvectors (main directions) of this tensor.

In high electric fields, the mobility along the main directions is proportional to the saturation velocity: $\mu_i \sim \frac{v_{\text{sat}}}{F}(1 + t_i)$. Therefore, the dependence of the saturation velocity on stress is similar to the mobility (with the factor $1 + t_i$). However, measured data shows that saturation velocity can have a different stress effect or no stress effect. Sentaurus Device has a calibration option to modify the dependency of saturation velocity on stress in the following form for the main directions:

$$v_{\text{sat}}^i = v_{\text{sat}}^0 \cdot \frac{(1 + \alpha \cdot t_i)}{(1 + t_i)} \quad (691)$$

where v_{sat}^0 is the stress-independent saturation velocity and α is a user-defined scalar factor. This factor can be specified using the keyword `SaturationFactor` in the `Piezo` section of the command file. For example:

```
Physics {
    Piezo( Model( Mobility(SaturationFactor = 0.5) ) )
}
```

This parameter can be specified separately for electrons and holes, for example:

```
Physics {
    Piezo( Model( Mobility(eSaturationFactor = 0.5 hSaturationFactor = 0) ) )
}
```

The default value for α is 1. This is equivalent to applying the stress enhancement factor to total mobility with $v_{\text{sat}}^i = v_{\text{sat}}^0$. Using $\alpha = 0$ with a Caughey–Thomas-type mobility (see [Eq. 261, p. 295](#)) is equivalent to applying the stress enhancement factor only to the low-field mobility, μ_{low} , with $v_{\text{sat}}^i = v_{\text{sat}}^0$.

23: Modeling Mechanical Stress Effect

Stress Tensor Applied to Low-Field Mobility

Figure 62 shows how the I_d - V_d curves depend on the parameter SaturationFactor.

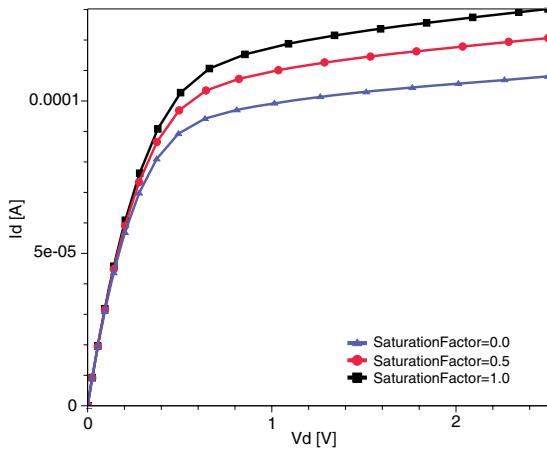


Figure 62 Dependency of I_d - V_d curves on SaturationFactor, with $V_g = 1.5$ V, and Stress = (0.6e9, 0, 0, 0, 0, 0, 0)

Stress Tensor Applied to Low-Field Mobility

In all the previous piezo models, the stress tensor factor was a linear factor applied to high-field mobility. Sentaurus Device allows also to apply the following diagonal mobility tensor factor to the low-field mobility:

$$\mu_{\text{high}} = \begin{bmatrix} \mu_{\text{high}}(\mu_{\text{low}} \cdot s_{xx}, \dots) & 0 & 0 \\ 0 & \mu_{\text{high}}(\mu_{\text{low}} \cdot s_{yy}, \dots) & 0 \\ 0 & 0 & \mu_{\text{high}}(\mu_{\text{low}} \cdot s_{zz}, \dots) \end{bmatrix} \quad (692)$$

where:

- s_{ii} are the diagonal elements of the stress tensor factor $\left(\bar{1} + \beta \frac{\Delta \mu_{\alpha}^{\text{Stress}}}{\mu_{\alpha}^0} \right)$, $\alpha = n, p$ in Eq. 689, p. 632. In this model, off-diagonal elements are not used.
- $\mu_{\text{high}}(\dots)$ is a scalar function that computes the high-field mobility (see High-Field Saturation on page 294).

This model can be specified in the Math section of the input command file as follows:

```
Math { StressMobilityDependence = TensorFactor }
```

In this case, the dependency of saturation velocity on stress is given by:

$$v_{\text{sat}}^i = v_{\text{sat}}^0 \cdot (1 + \alpha \cdot t_i) \quad (693)$$

which results in the same dependency on SaturationFactor described in [Dependency of Saturation Velocity on Stress on page 633](#) when a Caughey–Thomas-type mobility model is used.

The high-field mobility tensor (in [Eq. 692](#)) and the corresponding factors $\mu_{\text{high}}(\mu_{\text{low}} \cdot s_{ii}, \dots) / \mu_{\text{high}}(\mu_{\text{low}}, \dots)$ can be plotted on the mesh nodes. To plot these values for electron and hole mobilities, use the following keywords in the Plot section:

- eTensorMobilityXX, eTensorMobilityYY, eTensorMobilityZZ
- hTensorMobilityXX, hTensorMobilityYY, hTensorMobilityZZ
- eTensorMobilityFactorXX, eTensorMobilityFactorYY,
eTensorMobilityFactorZZ
- hTensorMobilityFactorXX, hTensorMobilityFactorYY,
hTensorMobilityFactorZZ

Piezoelectric Polarization

Sentaurus Device provides two models (strain and stress) to compute polarization effects in GaN devices. They can be activated in the Physics section of the command file as follows:

```
Physics {
    Piezoelectric_Polarization (strain)
    Piezoelectric_Polarization (stress)
}
```

The piezoelectric polarization vector and the piezoelectric charge can be plotted by:

```
Plot {
    PE_Polarization/vector
    PE_Charge
}
```

Sentaurus Device also offers a corresponding PMI model (see [Piezoelectric Polarization on page 1013](#)).

23: Modeling Mechanical Stress Effect

Piezoelectric Polarization

Strain Model

This model is based on the work by Ambacher *et al.* [25][26]. It captures the first-order effect of polarization vectors in AlGaN/GaN HFETs: the interface charge induced due to the discontinuity in the vertical component of the polarization vector at material interfaces.

The polarization vector is computed as follows:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} P_x^{\text{sp}} \\ P_y^{\text{sp}} \\ P_z^{\text{sp}} + 2d_{31} \cdot \text{strain} \cdot (c_{11} + c_{12} - 2c_{13}^2/c_{33}) \end{bmatrix} \quad (694)$$

where P^{sp} denotes the spontaneous polarization vector [C/cm^2], d_{31} is a piezoelectric coefficient [cm/V], and c_{ij} are stiffness constants [Pa]. The value of strain is computed as:

$$\text{strain} = (1 - \text{relax}) \cdot (a_0 - a)/a \quad (695)$$

where a_0 represents the unstrained lattice constant [\AA], a is the strained lattice constant [\AA], and ‘relax’ denotes a relaxation parameter [1].

The quantities P^{sp} , d_{31} , and c_{ij} are defined in the crystal system. The polarization vector is first computed in crystal coordinates and is converted to simulation coordinates afterwards.

Stress Model

Although, in most practical situations, there are only in-plane stress components due to lattice mismatch, the vertical and shear stress components give rise to in-plane piezopolarization components, which lead to volume charge densities and, therefore, potential variations. The stress model computes the full polarization vector in tensor form without simplifying assumptions:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} P_x^{\text{sp}} \\ P_y^{\text{sp}} \\ P_z^{\text{sp}} \end{bmatrix} + \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} & d_{16} \\ d_{21} & d_{22} & d_{23} & d_{24} & d_{25} & d_{26} \\ d_{31} & d_{32} & d_{33} & d_{34} & d_{35} & d_{36} \end{bmatrix} \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{yz} \\ \sigma_{xz} \\ \sigma_{xy} \end{bmatrix} \quad (696)$$

where P^{sp} denotes the spontaneous polarization vector [C/cm^2], d_{ij} are the piezoelectric coefficients [cm/V], and:

$$\begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{yz} \\ \sigma_{xz} \\ \sigma_{xy} \end{bmatrix} \quad (697)$$

is the stress tensor [Pa].

The quantities P^{sp} and d_{ij} are defined in the crystal system. The stress tensor σ is defined in the stress system. It is first converted from stress coordinates to crystal coordinates. Then, the polarization vector is computed in crystal coordinates. Afterwards, the polarization vector is converted to simulation coordinates.

Poisson Equation

Based on the polarization vector, the piezoelectric charge is computed according to:

$$q_{\text{PE}} = -\nabla P \quad (698)$$

and is added to the right-hand side of the Poisson equation:

$$\nabla \epsilon \cdot \nabla \phi = -q(p - n + N_D - N_A + q_{\text{PE}}) \quad (699)$$

Only the first d components of the polarization vector are used to compute the polarization charge, where d denotes the dimension of the problem.

Parameter File

The following parameters can be specified in the parameter file. All of them are mole fraction dependent, except `a0` and `relax`:

```
Piezoelectric_Polarization {
    # piezoelectric coefficients [cm/V]
    d11 = ...
    d12 = ...
    d13 = ...
    d14 = ...
```

23: Modeling Mechanical Stress Effect

Piezoelectric Polarization

```
d15 = ...
d16 = ...
d21 = ...
d22 = ...
d23 = ...
d24 = ...
d25 = ...
d26 = ...
d31 = ...
d32 = ...
d33 = ...
d34 = ...
d35 = ...
d36 = ...

# spontaneous polarization [C/cm^2]
psp_x = ...
psp_y = ...
psp_z = ...

# stiffness constants [Pa]
c11 = ...
c12 = ...
c13 = ...
c33 = ...

# strain parameters
a0 = ... # [Å]
a = ... # [Å]
relax = ... # [1]
}
```

Coordinate Systems

The x-axis and y-axis of the simulation system are defined in the parameter file:

```
LatticeParameters {
    X = (1, 0, 0)
    Y = (0, 0, -1)
}
```

The z-axis is computed as the outer vector product of the x-axis and y-axis. The simulation system is defined relative to the crystal system. If the keyword `CrystalAxis` is present, the crystal system would be defined relative to the simulation system (see [Using Deformation Potential Model on page 599](#)).

In the example above, the x-axis of the simulation system coincides with the x-axis of the crystal system. The y-axis of the simulation system runs along the negative z-axis of the crystal system. This is a common definition for 2D simulations.

The x-axis and y-axis of the stress system are defined in the `Physics` section:

```
Physics {
    Piezo (
        OriKddX = (-0.96 0.28 0)
        OriKddY = (0.28 0.96 0)
    )
}
```

The z-axis is computed as the outer vector product of the x-axis and y-axis. The stress system is defined relative to the simulation system (see [Using Stress-dependent Models on page 629](#)).

References

- [1] J. Bardeen and W. Shockley, “Deformation Potentials and Mobilities in Non-Planar Crystals,” *Physical Review*, vol. 80, no. 1, pp. 72–80, 1950.
- [2] I. Goroff and L. Kleinman, “Deformation Potentials in Silicon. III. Effects of a General Strain on Conduction and Valence Levels,” *Physical Review*, vol. 132, no. 3, pp. 1080–1084, 1963.
- [3] J. J. Wortman, J. R. Hauser, and R. M. Burger, “Effect of Mechanical Stress on p-n Junction Device Characteristics,” *Journal of Applied Physics*, vol. 35, no. 7, pp. 2122–2131, 1964.
- [4] P. Smeys, *Geometry and Stress Effects in Scaled Integrated Circuit Isolation Technologies*, Ph.D. thesis, Stanford University, Stanford, CA, USA, August 1996.
- [5] M. Lades *et al.*, “Analysis of Piezoresistive Effects in Silicon Structures Using Multidimensional Process and Device Simulation,” in *Simulation of Semiconductor Devices and Processes (SISDEP)*, vol. 6, Erlangen, Germany, pp. 22–25, September 1995.
- [6] J. L. Egley and D. Chidambarrao, “Strain Effects on Device Characteristics: Implementation in Drift-Diffusion Simulators,” *Solid-State Electronics*, vol. 36, no. 12, pp. 1653–1664, 1993.
- [7] J. F. Nye, *Physical Properties of Crystals*, Oxford: Clarendon Press, 1985.
- [8] G. L. Bir and G. E. Pikus, *Symmetry and Strain-Induced Effects in Semiconductors*, New York: John Wiley & Sons, 1974.

23: Modeling Mechanical Stress Effect

References

- [9] E. Uengersboeck *et al.*, “The Effect of General Strain on the Band Structure and Electron Mobility of Silicon,” *IEEE Transactions on Electron Devices*, vol. 54, no. 9, pp. 2183–2190, 2007.
- [10] T. Manku and A. Nathan, “Valence energy-band structure for strained group-IV semiconductors,” *Journal of Applied Physics*, vol. 73, no. 3, pp. 1205–1213, 1993.
- [11] V. Sverdlov *et al.*, “Effects of Shear Strain on the Conduction Band in Silicon: An Efficient Two-Band $k \cdot p$ Theory,” in *Proceedings of the 37th European Solid-State Device Research Conference (ESSDERC)*, Munich, Germany, pp. 386–389, September 2007.
- [12] F. L. Madarasz, J. E. Lang, and P. M. Hemeier, “Effective masses for nonparabolic bands in *p*-type silicon,” *Journal of Applied Physics*, vol. 52, no. 7, pp. 4646–4648, 1981.
- [13] C. Y.-P. Chao and S. L. Chuang, “Spin-orbit-coupling effects on the valence-band structure of strained semiconductor quantum wells,” *Physical Review B*, vol. 46, no. 7, pp. 4110–4122, 1992.
- [14] M. V. Fischetti and S. E. Laux, “Band structure, deformation potentials, and carrier mobility in strained Si, Ge, and SiGe alloys,” *Journal of Applied Physics*, vol. 80, no. 4, pp. 2234–2252, 1996.
- [15] Z. Wang, *Modélisation de la piézorésistivité du Silicium: Application à la simulation de dispositifs M.O.S.*, Ph.D. thesis, Université des Sciences et Technologies de Lille, Lille, France, 1994.
- [16] Y. Kanda, “A Graphical Representation of the Piezoresistance Coefficients in Silicon,” *IEEE Transactions on Electron Devices*, vol. ED-29, no. 1, pp. 64–70, 1982.
- [17] S. Selberherr, *Analysis and Simulation of Semiconductor Devices*, Wien: Springer, 1984.
- [18] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer*, Redwood City, California: Addison-Wesley Publishing Company, 2nd ed., 1991.
- [19] K. Matsuda *et al.*, “Nonlinear piezoresistance effects in silicon,” *Journal of Applied Physics*, vol. 73, no. 4, pp. 1838–1847, 1993.
- [20] S. Dhar *et al.*, “Electron Mobility Model for Strained-Si Devices,” *IEEE Transactions on Electron Devices*, vol. 52, no. 4, pp. 527–533, 2005.
- [21] E. Uengersboeck *et al.*, “Physical Modeling of Electron Mobility Enhancement for Arbitrarily Strained Silicon,” in *11th International Workshop on Computational Electronics (IWCE)*, Vienna, Austria, pp. 141–142, May 2006.
- [22] B. Obradovic *et al.*, “A Physically-Based Analytic Model for Stress-Induced Hole Mobility Enhancement,” in *10th International Workshop on Computational Electronics (IWCE)*, West Lafayette, IN, USA, pp. 26–27, October 2004.

- [23] L. Smith *et al.*, “Exploring the Limits of Stress-Enhanced Hole Mobility,” *IEEE Electron Device Letters*, vol. 26, no. 9, pp. 652–654, 2005.
- [24] J. R. Watling, A. Asenov, and J. R. Barker, “Efficient Hole Transport Model in Warped Bands for Use in the Simulation of Si/SiGe MOSFETs,” in *International Workshop on Computational Electronics (IWCE)*, Osaka, Japan, pp. 96–99, October 1998.
- [25] O. Ambacher *et al.*, “Two-dimensional electron gases induced by spontaneous and piezoelectric polarization charges in N- and Ga-face AlGaN/GaN heterostructures,” *Journal of Applied Physics*, vol. 85, no. 6, pp. 3222–3233, 1999.
- [26] O. Ambacher *et al.*, “Two dimensional electron gases induced by spontaneous and piezoelectric polarization in undoped and doped AlGaN/GaN heterostructures,” *Journal of Applied Physics*, vol. 87, no. 1, pp. 334–344, 2000.

23: Modeling Mechanical Stress Effect

References

This chapter describes the model for carrier transport in magnetic fields.

Model Description

For analysis of magnetic field effects in semiconductor devices, the transport equations governing the flow of electrons and holes in the interior of the device must be set up and solved.

To this end, the commonly used drift-diffusion-based model of the carrier current densities \vec{J}_n and \vec{J}_p must be augmented by magnetic field-dependent terms that account for the action of the transport equations governing the flow of electrons and holes in the interior of the device, the *Lorentz force* on the motion of the carriers [1][2][3]:

$$\vec{J}_\alpha = \mu_\alpha \vec{g}_\alpha + \mu_\alpha \frac{1}{1 + (\mu_\alpha^* B)^2} [\mu_\alpha^* \vec{B} \times \vec{g}_\alpha + \mu_\alpha^* \vec{B} \times (\mu_\alpha^* \vec{B} \times \vec{g}_\alpha)] \quad \text{with } \alpha = n, p \quad (700)$$

where:

\vec{g}_α is current vector without mobility (see [Current Densities on page 577](#)).

μ_α^* is the Hall mobility.

\vec{B} is the magnetic induction vector, and B is the magnitude of this vector.

The perpendicular (transverse) components of Hall and drift mobility are related by $\mu_n^* = r_n \mu_n$ and $\mu_p^* = r_p \mu_p$, where r_n and r_p denote the Hall scattering factors. In the case of bulk silicon, typical values are $r_n = 1.1$ and $r_p = 0.7$.

24: Galvanic Transport Model

Using Galvanic Transport Model

Using Galvanic Transport Model

In the Physics section of the command file, specify the magnetic field vector using the keyword `MagneticField = (<x>, <y>, <z>)`. In the following example, a field of 0.1 Tesla is applied parallel to the z-axis:

```
Physics {...
    MagneticField = (0.0, 0.0, 0.1)
}
```

Numeric Computation of Current Vector Without Mobility

Sentaurus Device computes the current vector without mobility for each mesh element and then uses the box method (see [Chapter 33 on page 885](#)) for discretization of the continuity equation with the magnetic field.

This computation is based on the least squares method. Let a mesh element (a triangle in this example) contain the vertices v_0, v_1, v_2 . Let the projection of the current vector without mobility to the edges be equal to b_0, b_1, b_2 (these values are computed as Scharfetter–Gummel approximations), and the edge directions (unit vectors along edges) are equal to $\vec{a}_0, \vec{a}_1, \vec{a}_2$ (see [Figure 63](#)).

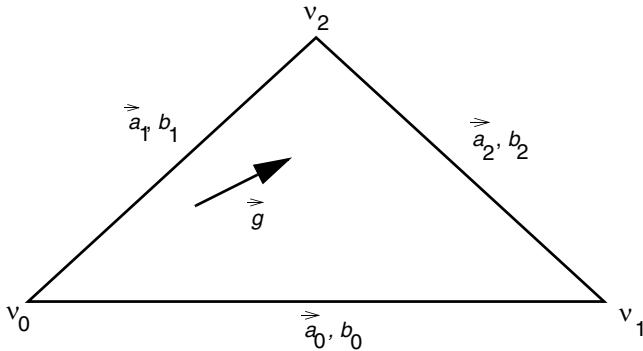


Figure 63 Vertices v_0, v_1, v_2 , edge direction $\vec{a}_0, \vec{a}_1, \vec{a}_2$, current projection b_0, b_1, b_2 , and current vector without mobility \vec{g}

Assume that the edge direction components are equal to $\vec{a}_i = (a_{i0}, a_{i1})$, matrix $A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \end{bmatrix}$, and vector $\vec{b} = (b_0, b_1, b_2)^T$.

Sentaurus Device computes the current vector without mobility as the solution of the following minimization problem:

$$\|A \cdot \vec{g} - \vec{b}\|^2 \rightarrow \text{minimum} \quad (701)$$

It is well known that this problem is equivalent to the linear system:

$$A^T A \cdot \vec{g} = A^T \cdot \vec{b} \quad (702)$$

and has the solution:

$$\vec{g} = (A^T A)^{-1} A^T \cdot \vec{b} \quad (703)$$

The computation for all other element types are similar.

NOTE The galvanic transport model cannot be combined with the following models: hydrodynamic, impact ionization, aniso, piezo, and quantum modeling.

NOTE The value of the magnetic field is a very critical parameter for convergence. For doping-dependent mobility, only the convergence is reliable up to $B = 10\text{ T}$. For more general mobility, the convergence has a limit of the magnetic field up to $B = 1\text{ T}$.

References

- [1] W. Allegretto, A. Nathan, and H. Baltes, “Numerical Analysis of Magnetic-Field-Sensitive Bipolar Devices,” *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 4, pp. 501–511, 1991.
- [2] C. Riccobene *et al.*, “Operating Principle of Dual Collector Magnetotransistors Studied by Two-Dimensional Simulation,” *IEEE Transactions on Electron Devices*, vol. 41, no. 7, pp. 1136–1148, 1994.
- [3] C. Riccobene *et al.*, “First Three-Dimensional Numerical Analysis of Magnetic Vector Probe,” in *IEDM Technical Digest*, San Francisco, CA, USA, pp. 727–730, December 1994.

24: Galvanic Transport Model

References

This chapter describes the models that are important for lattice heating and the thermodynamic model: heat capacity, thermal conductivity, and thermoelectric power.

Heat Capacity

[Table 100](#) lists the values of the heat capacity used in the simulator.

Table 100 Values of heat capacity c for various materials

Material	c [J/K cm ³]	Reference
Silicon	1.63	[1]
Ceramic	2.78	[2]
SiO ₂	1.67	[2]
Poly Si	1.63	≈ Si

By default, or when you specify `HeatCapacity(TempDep)` in the `Physics` section, the temperature dependency of the lattice heat capacity is modeled by the empirical function:

$$c_L = cv + cv_b T + cv_c T^2 + cv_d T^3 \quad (704)$$

The equation coefficients can be specified in the parameter file by using the syntax:

```
LatticeHeatCapacity{
    cv = 1.63          # [J/(K cm^3)]
    cv_b = 0.0000e+00  # [J/(K^2 cm^3)]
    cv_c = 0.0000e+00  # [J/(K^3 cm^3)]
    cv_d = 0.0000e+00  # [J/(K^4 cm^3)]
}
```

All these coefficients can be mole fraction-dependent for mole-dependent materials.

To use a PMI model to compute heat capacity, specify the name of the model as a string as an option to `HeatCapacity` (see [Heat Capacity on page 994](#)). To use a multistate configuration-dependent PMI for heat capacity, specify the model and its parameters as arguments to `PMIModel`, which, in turn, is an argument to `HeatCapacity` (see [Multistate Configuration-dependent Heat Capacity on page 996](#)).

25: Thermal Properties

Thermal Conductivity

To use a constant lattice heat capacity without touching the parameter file, specify `HeatCapacity(Constant)`.

Thermal Conductivity

Sentaurus Device uses the following temperature-dependent thermal conductivity κ in silicon [3]:

$$\kappa(T) = \frac{1}{a + bT + cT^2} \quad (705)$$

where $a = 0.03\text{ cmKW}^{-1}$, $b = 1.56 \times 10^{-3}\text{ cmW}^{-1}$, and $c = 1.65 \times 10^{-6}\text{ cmW}^{-1}\text{K}^{-1}$. The range of validity is from 200 K to well above 600 K. Values of the thermal conductivity for some materials are given in [Table 101](#).

Table 101 Values of thermal conductivity κ of silicon versus temperature

Material	κ [W/(cm K)]	Reference
Silicon	Eq. 705	[3]
Ceramic	0.167	[4]
SiO_2	0.014	[1]
Poly Si	1.5	\approx Si

As additional options to the standard specification of the thermal conductivity model, there are two different expressions to define either thermal resistivity $\chi = 1/\kappa$ or thermal conductivity for any material. This is performed by using `Formula` in the parameter file or special keywords in the command file.

For `Formula=0` (thermal resistivity specification), Sentaurus Device uses:

$$\chi = 1/\kappa + 1/\kappa_b T + 1/\kappa_c T^2 \quad (706)$$

For `Formula=1` (thermal conductivity specification), it is:

$$\kappa = \kappa + \kappa_b T + \kappa_c T^2 \quad (707)$$

Using the following syntax in the parameter file, the expressions can be switched coefficients specified:

```
Kappa{
    Formula = 0
    1/kappa = 0.03          # [K cm/W]
    1/kappa_b = 1.5600e-03  # [cm/W]
    1/kappa_c = 1.6500e-06  # [cm/(W K)]
    kappa = 1.5             # [W/(K cm)]
    kappa_b = 0.0000e+00    # [W/(K^2 cm)]
    kappa_c = 0.0000e+00    # [W/(K^3 cm)])
}
```

The Physics section of the command file provides more flexibility to switch these expressions by using the keywords:

```
ThermalConductivity(
    TempDep Conductivity # Formula = 1
    Constant Conductivity # Formula = 1 without temperature dependence
    TempDep Resistivity # Formula = 0
    Constant Resistivity # Formula = 0 without temperature dependence
)
```

By default, Sentaurus Device uses `Formula` specified in the parameter file. All these coefficients in the parameter file can be mole dependent for mole-dependent materials.

Furthermore, a simple PMI and a multistate configuration-dependent PMI are available to compute thermal conductivity. See [Thermal Conductivity on page 989](#) and [Multistate Configuration-dependent Thermal Conductivity on page 991](#) for details.

Thermoelectric Power (TEP)

Theoretically, the electron and hole absolute thermoelectric powers P_n and P_p for nondegenerate semiconductors can be written as [5][6]:

$$P_n = -\kappa_n \frac{k}{q} \left[\left(\frac{5}{2} - s_n \right) + \ln \left(\frac{N_C}{n} \right) \right] \quad (708)$$

$$P_p = \kappa_p \frac{k}{q} \left[\left(\frac{5}{2} - s_p \right) + \ln \left(\frac{N_V}{p} \right) \right] \quad (709)$$

To use these expressions, specify `AnalyticTEP` in the `Physics` section of the command file. The coefficients in these equations are available in the `TEPower` parameter set (see [Table 102 on page 650](#)).

25: Thermal Properties

References

Table 102 Coefficients for thermoelectric power

Symbol	Parameter name	Default
κ_n	scale_n	1
κ_p	scale_p	1
s_n	s_n	1
s_p	s_p	1

Without the keyword `AnalyticTEP`, Sentaurus Device uses a table of experimental values of the TEPs for silicon published by Geballe and Hull [7] as a function of temperature and carrier concentration. Sentaurus Device extrapolates $P_n T$ and $P_p T$ linearly for temperatures between 360 K and 500 K, thereby preserving the $1/T$ dependence of data presented at higher temperatures by Fulkerson *et al.* [8], which holds up to near the intrinsic temperature.

P_n and P_p are shown in Figure 64 as a function of temperature and carrier concentration as used in Sentaurus Device.

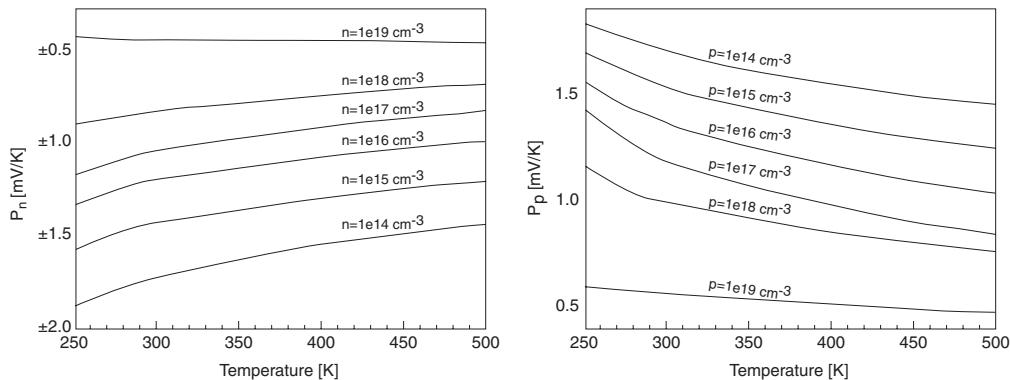


Figure 64 TEPs P_n (left) and P_p (right) as a function of temperature and carrier concentration

References

- [1] S. M. Sze, *Physics of Semiconductor Devices*, New York: John Wiley & Sons, 2nd ed., 1981.
- [2] D. J. Dean, *Thermal Design of Electronic Circuit Boards and Packages*, Ayr, Scotland: Electrochemical Publications Limited, 1985.
- [3] C. J. Glassbrenner and G. A. Slack, “Thermal Conductivity of Silicon and Germanium from 3°K to the Melting Point,” *Physical Review*, vol. 134, no. 4A, pp. A1058–A1069, 1964.

- [4] S. S. Furkay, "Thermal Characterization of Plastic and Ceramic Surface-Mount Components," *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, vol. 11, no. 4, pp. 521–527, 1988.
- [5] R. A. Smith, *Semiconductors*, Cambridge: Cambridge University Press, 2nd ed., 1978.
- [6] C. Herring, "The Role of Low-Frequency Phonons in Thermoelectricity and Thermal Conduction," in *Semiconductors and Phosphors: Proceedings of the International Colloquium*, Garmisch-Partenkirchen, Germany, pp. 184–235, August 1956.
- [7] T. H. Geballe and G. W. Hull, "Seebeck Effect in Silicon," *Physical Review*, vol. 98, no. 4, pp. 940–947, 1955.
- [8] W. Fulkerson *et al.*, "Thermal Conductivity, Electrical Resistivity, and Seebeck Coefficient of Silicon from 100 to 1300°K," *Physical Review*, vol. 167, no. 3, pp. 765–782, 1968.

25: Thermal Properties

References

Part III Physics of Lasers and Light-Emitting Diodes

This part of the Sentaurus Device manual contains the following chapters:

[Chapter 26 Introduction to Lasers and Light-emitting Diodes on page 655](#)

[Chapter 27 Lasers on page 687](#)

[Chapter 28 Light-emitting Diodes on page 721](#)

[Chapter 29 Optical Mode Solver for Lasers on page 761](#)

[Chapter 30 Modeling Quantum Wells on page 803](#)

[Chapter 31 Additional Features of Laser or LED Simulations on page 855](#)

Introduction to Lasers and Light-emitting Diodes

This chapter is an introduction to the simulation of lasers and light-emitting diodes in Sentaurus Device.

Sentaurus Device includes optional models for the comprehensive simulation of semiconductor lasers and light-emitting diodes (LEDs). Both edge-emitting lasers and vertical-cavity surface-emitting lasers (VCSELs) are supported. Drift-diffusion or hydrodynamic transport equations for the carriers, the Schrödinger equation for quantum well gain, modal optical rate equations, and the Helmholtz equation are solved self-consistently in the quasistationary and transient modes.

Spontaneous and stimulated optical recombinations are calculated in the active and bulk regions according to Fermi's golden rule. They are added as carrier recombination mechanisms in the continuity equations and as modal gain and spontaneous emission in the photon rate equations. Different line-width broadening models are available for gain broadening.

Both bulk and quantum well (QW) lasers can be simulated. In the case of QW lasers, the optical polarization dependence of the optical matrix element is automatically taken into account. The QW subbands are calculated as the solution of the single-band Schrödinger equation in the effective mass approximation, assuming a box-shaped potential or by a multiple-bands k.p method. Both zinc-blende and wurtzite crystal structures can be treated. Strain effects can also be taken into account. The distribution of carriers in the well is determined according to the quantum mechanical wavefunctions and QW density-of-states.

In the unquantized direction, drift-diffusion transport applies to the carriers. In the quantized direction, an advanced transport model separates the QW carrier distributions into a bound and continuum distribution. The fraction of bound and continuum carriers is then self-consistently computed by additional scattering equations, mainly contributed by carrier-carrier and carrier-optical phonon scatterings.

The next section is a short introduction on how to set up a laser simulation with single and dual grids, and which type of output can be extracted from the simulation. The theoretical foundations of the laser simulator with detailed equations are discussed, followed by a description of other useful features of the laser and LED simulator. Finally, there is a discussion on how to simulate different types of laser and LED.

Overview

A laser or an LED simulation is different from a silicon device simulation in three aspects:

- The Helmholtz equation must be solved to determine the optics of the device.
- A photon rate equation must be included to couple the electronics to the optics.
- The carrier-scattering processes at the quantum well must be treated differently because it is no longer in the drift-diffusion regime. In this case, a new syntax must be introduced to activate the laser simulation within the Sentaurus Device framework. The laser-specific and LED-specific extensions of the command file are examined in the next sections.

The procedure for performing a laser simulation is similar to that for a silicon device simulation. [Figure 65](#) illustrates the procedure.

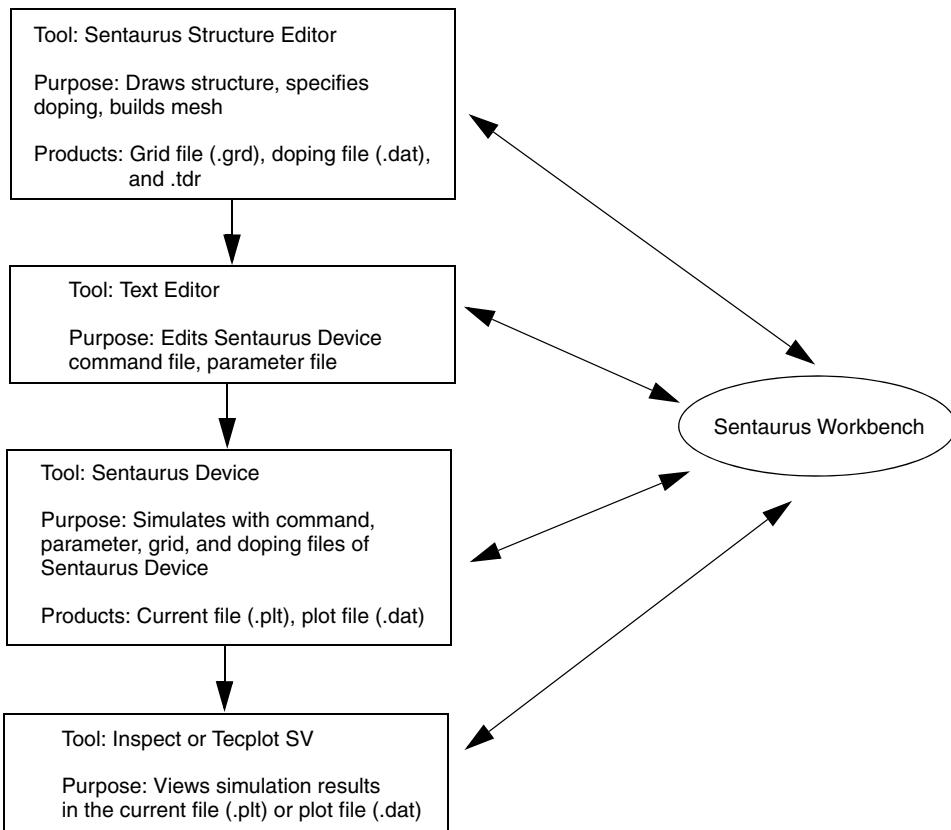


Figure 65 Flow of tasks in a single laser simulation run

First, the structure and doping profile are drawn in Sentaurus Structure Editor (refer to the *Sentaurus Structure Editor User Guide*). Second, Sentaurus Structure Editor generates the mesh and the doping information on the mesh, and saves them in the grid file (with extension .grd) and doping file (with extension .dat), respectively. Instead of drawing the structure laboriously, you have the option to write a script to construct the structure, and build the mesh and doping automatically. In this way, it is possible to parameterize any device dimension, material composition, and doping profile easily. This feature is used by Sentaurus Workbench to control batch jobs and automatically optimize user-specified performance benchmarks through parameter variation.

Next, you can use any editor to edit the command file and parameter file. The parameter file contains a complete list of default material parameter models such as band gap, thermal conductivities, recombination parameters, traps parameters, effective masses, and mobilities. You can edit any of these material parameters. The command file contains instructions to run the simulation. If you run a 1D, 2D, or 3D simulation, only the grid and doping files change. The command file and parameter file are unchanged.

After the simulation is completed, Sentaurus Device saves the final results in the current file (with extension .plt) and the plot file (with extension .dat). These results can be viewed and plotted in Inspect and Tecplot SV, respectively. If you specify some other feature such as far-field computation, additional files are produced that can be viewed in Inspect or Tecplot SV, depending on the file type.

All the abovementioned steps can be controlled from Sentaurus Workbench, which can also schedule jobs over the network and extract specified results of the simulation. In this manner, automation can be achieved for device optimization. Refer to the *Sentaurus Workbench User Guide* for more information.

Command File Syntax

The command file permits full control of how a simulation will run. This includes choosing the type of numeric solver, the physics to be included, and the equations to couple. This fundamental framework provides a platform with unlimited expansion possibilities for new features, new models, and new devices. The fastest way to understand the command file syntax is to look at a generic example that you can copy and modify for your specific needs. Two generic examples are presented for typical laser and LED simulations.

NOTE The character '#' in the command file means that any text proceeding from '#' is treated as a remark or as a preprocessor directive.

Simulating Single-Grid Edge-emitting Laser

This generic example uses the same grid for both the optical and electrical problems. The optical problem is defined by the optics solver and the electrical problem is defined by the semiconductor transport equations.

```
File {
    Grid      = "mesh_mdr.grd"
    Doping    = "mesh_mdr.dat"
    Parameters = "des_las.par"

    Current = "current"
    Output   = "output"
    Plot     = "plot"

    SaveOptField = "laserfield" # save optics
    FarField     = "farfield"    # farfield computation
    Gain         = "gain"        # save gain curves
}

Electrode {
    {Name="p_Contact" voltage=0.8 AreaFactor = 200}
    {Name="n_Contact" voltage=0.0 AreaFactor = 200}
}

Plot {
    # ----- Extra variables for laser simulation -----
    QWeDensity QWhDensity
    QWeQuasiFermi QWhQuasiFermi
    MatGain
    LaserIntensity
    OpticalIntensityMode0
    OpticalIntensityMode1
    OpticalPolarizationAngleMode0
    OpticalPolarizationAngleMode1
}

Physics {
    AreaFactor=2    # for symmetric simulation
    Laser (
        Optics (
            FEScalar (
                EquationType = Waveguide
                Symmetry = Symmetric           # or NonSymmetric or Periodic
                LasingWavelength = 800          # [nm]
                TargetEffectiveIndex = (3.4 3.34) # initial guess
                Polarization = (TE TM)        # for scalar solver only
            )
        )
    )
}
```

```

        Boundary = ("Type2" "Type1")      # choose boundary conditions
        ModeNumber = 2                  # up to 10 modes
    )
)

TransverseModes          # for edge emitter simulation
CavityLength = 900        # [micrometer]
lFacetReflectivity = 0.9 # Left facet power reflectivity
rFacetReflectivity = 0.4 # Right facet power reflectivity
OpticalLoss = 10.0        # [1/cm]
WaveguideLoss            # activate feedback of loss from Optics

Broadening (Type=Lorentzian Gamma=0.010) # Gamma in [eV]

# ----- Specify QW physics -----
qwTransport
qwExtension = AutoDetect # auto read of QW widths
qwScatmodel
QWeScatTime = 8e-13     # [s]
QWhScatTime = 4e-13     # [s]
eQWMobility = 9200       # [cm^2/Vs]
hQWMobility = 400         # [cm^2/Vs]

Strain          # include QW Strain effects
StimScaling = 1.0
SponScaling = 1.0
RefractiveIndex (TemperatureDep CarrierDep)
)

# ----- Specify transport physics -----
Thermionic          # thermionic emission over interfaces
HeteroInterface      # discontinuous bandgap & quasi-Fermi level
Mobility ( DopingDependence )
Recombination ( SRH Auger )
EffectiveIntrinsicDensity ( NoBandGapNarrowing )
Fermi
}

Physics (region="pbulk") { MoleFraction(xfraction=0.28) }
Physics (region="nbulk") { MoleFraction(xfraction=0.28) }
Physics (region="psch") { MoleFraction(xfraction=0.09) }
Physics (region="nsch") { MoleFraction(xfraction=0.09) }

Physics (region="qwl") {
    MoleFraction( xfraction = 0.8 Grading=0.00 )
    Active                      # keyword to indicate QW region
}

```

26: Introduction to Lasers and Light-emitting Diodes

Command File Syntax

```
Math {
    Digits = 5
    Extrapolate
    Method = pardiso
    Iterations = 30
    Notdamped = 50
    RelErrControl
}

GainPlot { Range=(1.22,1.32) Intervals=150 }
FarFieldPlot { Range=(40,60) Intervals=40 Scalar1D Scalar2D Vector2D }

Solve {
    Poisson
    Coupled { Poisson Electron Hole }
    Coupled { Poisson Electron Hole QWhScatter QWeScatter PhotonRate }

    Quasistationary (
        InitialStep = 0.001
        MaxStep = 0.05
        Minstep = 1e-5

        Plot {Range=(0,1) Intervals=1}
        SaveOptField {Range=(0,1) Intervals=1}
        PlotFarField {Range=(0,1) Intervals=1}
        PlotGain {Range=(0,1) Intervals=1}

        Goal { name="p_Contact" Voltage=1.8 }

    ) {
        Plugin (BreakOnFailure) {
            Coupled { Poisson Electron Hole QWeScatter QWhScatter PhotonRate }
            Optics
            Wavelength
        }
    }
}
```

To elaborate on some remarks in this example of code:

- The general skeleton for the command file contains the `Electrode`, `File`, `Plot`, `Math`, `Physics`, and `Solve` sections. Within each section, additional subsections can be defined and, therefore, provide an object-oriented approach for defining the various parameters of the device and simulation.
- In the `File` section, the keyword of a specific option must be included to activate that option. For example, Sentaurus Device will only save the optical field if `SaveOptField` is included.

- In the `Electrode` section, `AreaFactor` gives the scaling factor to account for the device width in the longitudinal direction (1 μm by default).
- In the `Plot` section, a more complete list of the standard variables of Sentaurus Device can be found in [Plot Section on page 58](#). The laser simulation-specific plot variables are listed in the following sections.
- In the `Physics-Laser` section, `Laser` and `LED` simulations share these common options. The main difference between the different types of laser simulation is the choice of optical solver. In this example, the `FEScalar` (finite-element scalar) solver is used. Other optical solver choices include `FEVectorial`, `RayTrace`, `TMM1D`, and `EffectiveIndex`. The different types of optical solver are described in [Chapter 29 on page 761](#).
- In the `Physics-Laser` section, you can select different gain-broadening functions. The gain-broadening functions, nonlinear gain saturation model, and quantum well parameters are described in [Chapter 30 on page 803](#).
- In the `Physics` section, there are options to switch on the thermodynamic or hydrodynamic systems. To solve for the temperatures, it is necessary to define the `thermode` and couple the relevant equation in the command file. (This is not shown in this example.) Details of how to do this are in [Performing Temperature Simulation on page 859](#).
- In the `Math` section, there is a selection of numeric engines including `pardiso`, `ils`, `super`, `umf`, `slip`. You are encouraged to use different engines to find out which is the fastest and most accurate for your applications (see [Math Section on page 91](#)).
- The parameters to control the resolution of the gain plot and the far field are specified in the `GainPlot` and `FarFieldPlot` sections, respectively. These sections are at the same level as the `Physics`, `Math`, and `Solve` sections.
- In the `Solve-Quasistationary` section, the step size is expressed as a number between 0 and 1, which is mapped to the actual voltage ramping goal. In this case, the device was initially at 0.8 V and the goal was set to 1.8 V. Therefore, the unitless step of [0,1] maps to [0.8,1.8] V in the voltage ramp.
- In the `Solve-quasistationary` section, the keyword `Coupled` ensures that the listed equations are solved using Newton iterations. To enforce self-consistency of other systems (in this case, the `Wavelength` and `Optics`) in a Gummel-type iteration scheme, the `Plugin` feature was used. If it is not expected that the wavelength or optics will change as a function of the bias, they can be commented out.

Simulating Dual-Grid Edge-emitting Laser

In some cases, you may require different mesh sizes for the optical and electrical problems. As a general rule, the discretization for the optical mesh should be at least 20 points per wavelength for an accurate solution, but such fine discretization may not be necessary for the electrical problem.

In this case, the dual-grid capability is invoked. This capability is derived from the circuit mixed-mode feature of Sentaurus Device where the coupling of a few devices can be simulated in a self-consistent manner. The syntax of how this is performed is:

```
File {
    Output = "dual_log"
}

Math {
    # ----- Differences in a dual grid -----
    NoAutomaticCircuitContact
    DirectCurrent
    Method = blocked
    Submethod = pardiso
    # ----- The rest are the same as the single grid -----
    Digits = 5
    Extrapolate
    Iterations = 30
    Notdamped = 50
    RelErrControl
    # Cylindrical      # for VCSEL
}

* ----- Define the optical device ----- *
OpticalDevice optgrid {
    File {
        Grid = "optmesh_mdr.grd"
        Doping = "optmesh_mdr.dat"
        Parameters = "des_las.par"
    }

    Plot {
        LaserIntensity
        OpticalIntensityMode0
        OpticalIntensityMode1
    }

    Physics (region="pbulk") { MoleFraction(xfraction=0.28) }
    Physics (region="nbulk") { MoleFraction(xfraction=0.28) }
    Physics (region="psch") { MoleFraction(xfraction=0.09) }
}
```

```

Physics (region="nsch") { MoleFraction(xfraction=0.09) }

Physics (region="qw1") {
    MoleFraction( xfraction = 0.8 Grading=0.0 )
    Active
}
}

* ----- Define the electrical device ----- *
Device electricaldev {
    Electrode {
        {Name="p_Contact" voltage=0.8 AreaFactor = 200}
        {Name="n_Contact" voltage=0.0 AreaFactor = 200}
    }
}

File {
    Grid      = "elecmesh_mdr.grd"
    Doping    = "elecmesh_mdr.dat"
    Parameters = "des_las.par"

    Current   = "elec_current"
    Plot       = "elec_plot"
    SaveOptField = "laserfield"
    FarField   = "farfield"
    Gain       = "gain"
    # VCSELNearField = "vcselfield"  # for VCSEL
}

Plot {
    # ----- Extra variables for laser simulation -----
    QWeDensity QWhDensity
    QWeQuasiFermi QWhQuasiFermi
    MatGain
}

Physics {
    AreaFactor=2  # for symmetric simulation
    Laser (
        Optics (
            FEVectorial (
                EquationType = Waveguide          # or Cavity for VCSEL
                Symmetry = Symmetric           # for EEL only
                LasingWavelength = 800
                TargetEffectiveIndex = (3.4 3.32) # for EEL only
                Boundary = ("Type2" "Type1")     # for EEL only
                ModeNumber = 2
                # TargetWavelength = (850.4 851.7) # for VCSEL
                # AzimuthalExpansion = (0 1)      # for VCSELS

```

26: Introduction to Lasers and Light-emitting Diodes

Command File Syntax

```
        # Coordinates = Cylindrical          # Cylindrical for VCSEL
    )
)
# VCSEL()           # for VCSEL
TransverseModes      # for EEL
CavityLength = 900    # [micrometer], for EEL
lFacetReflectivity = 0.9 # Left facet power reflectivity, for EEL
rFacetReflectivity = 0.4 # Right facet power reflectivity, for EEL
OpticalLoss = 10.0     # [1/cm]
WaveguideLoss         # activate feedback of loss from Optics
Broadening (Type=Lorentzian Gamma=0.10) # [eV]

# ----- Specify QW physics -----
qwTransport
qwExtension = AutoDetect  # auto read QW widths
qwScatmodel
QWeScatTime = 8e-13   # [s]
QWhScatTime = 4e-13   # [s]
eQWMobility = 9200     # [cm^2/Vs]
hQWMobility = 400      # [cm^2/Vs]

Strain           # include QW Strain effects
StimScaling = 1.0
SponScaling = 1.0
RefractiveIndex (TemperatureDep CarrierDep)
)

# ----- Specify transport physics -----
Thermionic          # thermionic emission over interfaces
HeteroInterface      # discontinuous bandgap & quasi-Fermi level
Mobility ( DopingDependence )
Recombination ( SRH Auger )
EffectiveIntrinsicDensity ( NoBandGapNarrowing )
Fermi
}

Physics (region="pbulk") { MoleFraction(xfraction=0.28) }
Physics (region="nbulk") { MoleFraction(xfraction=0.28) }
Physics (region="psch") { MoleFraction(xfraction=0.09) }
Physics (region="nsch") { MoleFraction(xfraction=0.09) }

Physics (region="qw1") {
    MoleFraction( xfraction=0.8 Grading=0.0 )
    Active                                # keyword to indicate QW region
}

GainPlot      { Range=(1.22,1.32) Intervals=150 }
FarFieldPlot { Range=(40,60)      Intervals=40 Scalar1D Scalar2D Vector2D }
```

```
# VCSELNearFieldPlot { Position=0 Radius=10.0 NumberOfPoints=50 } # VCSEL
}

System {
    # --- Define d2 of type optgrid ---
    optgrid d2 ()
    # --- Define d1 of type diode, and coupled to d2 ---
    diode d1 (p_Contact=vdd n_Contact=gnd) {Physics{OptSolver="d2"}}
    # --- Set the initial bias voltage to circuit contacts ---
    Vsource_pset drive(vdd gnd) { dc=0.8 }
    Set ( gnd = 0.0 )
}

Solve {
    Poisson
    Coupled { Poisson Electron Hole }
    Coupled { Poisson Electron Hole QWhScatter QWeScatter PhotonRate
        Contact Circuit }

    Quasistationary (
        InitialStep = 0.001
        MaxStep = 0.05
        Minstep = 1e-5

        Plot {Range=(0,1) Intervals=1}
        SaveOptField {Range=(0,1) Intervals=1}
        PlotFarField {Range=(0,1) Intervals=1}
        PlotGain {Range=(0,1) Intervals=5}
        # VCSELNearField {Range=(0,1) Intervals=1}    # for VCSEL

        Goal { Parameter=drive.dc Value=1.6 }
    ) {
        Plugin (BreakOnFailure) {
            Coupled { Poisson Electron Hole QWeScatter QWhScatter PhotonRate
                Contact Circuit }
            Optics
            Wavelength
        }
    }
}
```

26: Introduction to Lasers and Light-emitting Diodes

Command File Syntax

This example uses the same device structure as in [Simulating Single-Grid Edge-emitting Laser on page 658](#), so that you can compare the single-grid and dual-grid simulations more clearly. Some important differences and similarities are:

- The optical and electrical grids may have different material-region physics, but the structural shape should be the same. This is to ensure that the interpolating of parameters and variables between the two grids is accurate. In this example, the electrical and optical devices have the same material regions, but the meshing resolutions are different, leading to two different grids.
- The control of the laser physics is encompassed in the definition of the electrical grid or device. The keyword to define the electrical grid or device is `Device`; `OpticalDevice` defines the optical grid.
- In the `System` section, `d2` is defined as a device instance of type `optgrid` and `d1` is defined as a device instance of type `diode`. The optical device instance `d2` is then coupled as a circuit element to the electrical device instance `d1` by specifying `OptSolver="d2"`. In addition, `p_Contact` and `n_Contact` of the electrical device are given the new names of `vdd` and `gnd`, respectively, in this coupled circuit (see [System Section on page 149](#) for more information about the `System` command).
- In the `Solve` section, the keywords `Contact` and `Circuit` in the `Coupled` statement ensure that self-consistency between the optical and electrical devices in the circuit is imposed.

Simulating Vertical-Cavity Surface-emitting Laser

A vertical-cavity surface-emitting laser (VCSEL) simulation is only performed if the keyword `VCSEL` is included in the `Physics-Laser` section of the command file:

```
# ----- Sentaurus Device command file for dual grid VCSEL simulation -----
File {
    Output = "dual_log"
}

# ----- Control of the numerical method in the mixed-mode circuit -----
Math {
    NoAutomaticCircuitContact
    DirectCurrent
    Method = blocked
    Submethod = pardiso
    Digits = 5
    Extrapolate
    ErReff(electron) = 1.e3
    ErReff(hole) = 1.e3
    Iterations = 30
    Notdamped = 50
    RelErrControl
```

```
}
```

```
# ===== Define the Optical grid =====
# ----- Use keyword OpticalDevice -----
OpticalDevice optgrid {

    File {
        # ----- Read in the optical mesh -----
        Grid = "optmesh_mdr.grd"
        Doping = "optmesh_mdr.dat"
        Parameters = "des_las.par"
    }
    Plot {
        LaserIntensity
        OpticalIntensityMode0
    }
    # ----- Material region physics for the optical problem -----
    ... # all the layers including the DBR layers

    # ----- Quantum wells -----
    Physics (region="qw1") {
        MoleFraction(xfraction = 0.8 Grading=0.00)
        Active
    }
    Physics (region="barr") { MoleFraction(xfraction=0.09) }
    Physics (region="qw2") {
        MoleFraction(xfraction = 0.8 Grading=0.00)
        Active
    }
}

# ===== End of Optical grid definition =====

# ===== Define the electrical grid and solver info =====
# ----- Use keyword Device-----
Device electricaldev {

    Electrode {
        { Name="p_Contact" voltage=0.8 AreaFactor=1 }
        { Name="n_Contact" voltage=0.0 AreaFactor=1 }
    }
    File {
        # ----- Read in electrical grid mesh -----
        Grid = "elecmesh_mdr.grd"
        Doping = "elecmesh_mdr.dat"
        Parameters = "des_las.par"
        Current = "elec_current"
        Plot =     "elec_plot"
    }
}
```

26: Introduction to Lasers and Light-emitting Diodes

Command File Syntax

```
SaveOptField = "laserfield"
Gain = "gain"
VCSELNearField = "nf"
FarField = "far"
}
Plot {
    # ----- Can include a long list -----
    LaserIntensity
    OpticalIntensityMode0
}
Physics {
    AreaFactor = 1
    # ----- Laser definition -----
    Laser(
        Optics(
            FEVectorial(EquationType = Cavity
                Coordinates = Cylindrical
                TargetWavelength = (781.2 776.5) # initial guesses [nm]
                TargetLifetime = (2.4 1.2) # initial guesses [ps]
                AzimuthalExpansion = (1 0) # cylindrical harmonic order
                ModeNumber = 2
            )
        )
    )
    # ----- Signify this is a VCSEL simulation -----
    VCSEL()
    OpticalLoss = 10.0          # [1/cm]
    # ----- Choose Gain broadening -----
    Broadening (Type=Lorentzian Gamma=0.10)      # [eV]
    # ----- Specify QW parameters -----
    qwTransport
    qwExtension = AutoDetect      # auto read QW widths
    qwScatmodel
    QWeScatTime = 8e-13          # [s]
    QWhScatTime = 4e-13          # [s]
    eQWMobility = 9200           # [cm^2/Vs]
    hQWMobility = 400            # [cm^2/Vs]
    # ----- QW Strain effects -----
    Strain
    SplitOff = 0.34              # [eV]
    # ----- can scale stim and spon gain independently -----
    StimScaling = 1.0
    SponScaling = 1.0
    # ----- Specify dependency of refractive index -----
    RefractiveIndex(TemperatureDep CarrierDep)
)
# ----- Specify transport physics -----
Thermionic
HeteroInterface
```

```

        Mobility ( DopingDependence )
        Recombination ( SRH Auger )
        EffectiveIntrinsicDensity ( NoBandGapNarrowing )
        Fermi
    }
# ----- Material region physics for electrical problem -----
# --- Simplify the DBR layers by an equivalent bulk region ---
Physics (region="pDBR_equivalent") { MoleFraction(xfraction=0.35) }
Physics (region="nDBR_equivalent") { MoleFraction(xfraction=0.35) }
Physics (region="psch") { MoleFraction(xfraction=0.09) }
Physics (region="nsch") { MoleFraction(xfraction=0.09) }
# ----- Quantum wells -----
Physics (region="qw1") {
    MoleFraction(xfraction = 0.8 Grading=0.00)
    Active
}
Physics (region="barr") { MoleFraction(xfraction=0.09) }
Physics (region="qw2") {
    MoleFraction(xfraction = 0.8 Grading=0.00)
    Active
}

# ----- Control the numerical method in the electrical problem -----
Math {
    Digits = 7
    # ----- Need to include this to specify cylindrical symmetry -----
    Cylindrical
}
# ----- Specify plot parameters -----
GainPlot { range=(1.22,1.32) intervals=150 }
FarFieldPlot { range=(80,80) intervals=50 Scalar1D Vector2D }
VCSELNearFieldPlot { Position=0 Radius=10.0 NumberOfPoints=50 }

# ===== End of electrical grid definition =====

# ===== Define the circuit mixed-mode system =====
System {
    # ----- Define opt1 of type optgrid -----
    optgrid opt1 ()
    # ----- Define d1 of type electricaldev, and coupled to opt1 -----
    electricaldev d1 (p_Contact=vdd n_Contact=gnd) {Physics{OptSolver="opt1"}}
    # ----- Set the initial bias voltage to circuit contacts -----
    Vsource_pset drive(vdd gnd){ dc=0.8 }
    Set ( gnd=0.0 )
}

# ===== Solver part =====
Solve {

```

26: Introduction to Lasers and Light-emitting Diodes

Command File Syntax

```
Poisson
coupled { Hole Electron Poisson Contact Circuit }
coupled { Hole Electron QWhScatter QWeScatter Poisson Contact Circuit }
coupled { Hole Electron QWhScatter QWeScatter Poisson Contact Circuit
          PhotonRate }

quasistationary (
    InitialStep = 0.001
    MaxStep = 0.01
    Minstep = 1e-7
    # ----- Plot and save various quantities at specified intervals of bias -
    -----
    Plot { range=(0,1) intervals=5 }
    PlotGain { range=(0,1) intervals=5 }
    SaveOptField { range=(0,1) intervals=3 }
    VCSELNearField { range=(0,1) intervals=3 }
    PlotFarField { range=(0,1) intervals=2 }
    # ----- Specify final voltage -----
    Goal { Parameter=drive.dc Value=1.6 })
{
    Plugin (BreakOnFailure) {
        Coupled { Electron Hole Poisson Contact Circuit QWeScatter
                  QWhScatter PhotonRate }
        Optics
        Wavelength
    }
}
}
```

Compare this VCSEL example with the edge-emitting laser example in [Simulating Dual-Grid Edge-emitting Laser on page 662](#), to highlight the syntaxes required for a VCSEL simulation. Some comments about the above example are:

- The vectorial optical solver (`FEVectorial`) has been used. To use the scalar solvers, you need only replace the `FEVectorial` section with the `EffectiveIndex` or `TMM1D` section.
- Dual-grid mixed-mode simulation is required for VCSELs regardless of whether the vectorial or scalar optical solvers are chosen.
- In the `Math` section for the electrical device, the keyword `Cylindrical` must be included so that the mesh volumes and areas in the finite box integration method are computed with cylindrical symmetry in mind.

Simulating Light-emitting Diode

An LED simulation is best run with a dual grid approach. The electrical grid must be dense in vicinities where carrier transport details are important. On the other hand, a coarse grid is needed for raytracing. Following is a skeletal command file syntax for a dual-grid LED simulation. Only highlights of the syntax that are important to LED simulations have been included. Missing details of the syntax (indicated by ellipses) can be completed by referring to the examples in [Simulating Dual-Grid Edge-emitting Laser on page 662](#) and [Simulating Light-emitting Diode on page 671](#).

The typical command file syntax is:

```

File { ... }

Math {
    # Choose to use Gaussian quadrature integration in gain calculations
    BroadeningIntegration ( GaussianQuadrature ( Order = 10 ) )
    SponEmissionIntegration ( GaussianQuadrature ( Order = 5 ) )
    Number_Of_Threads = 2           # Choose multithreading for ray tracer
    StackSize = 20000000          # Set stack size to 20MB
}

#####
### Define the optical solver part here:
#####
OpticalDevice OSolver {
    File {
        Grid      = "optic_white_msh.grd"
        Doping    = "optic_white_msh.dat"
        Parameters = "optparafile.par"
    }

    # These raytrace contacts are only defined in the optical grid
    RayTraceBC {
        { Name = "qwlphotonrecycle"          # full photon recycle contact
          fullphotonrecycle = reemit(0.2)
        }
        { Name = "qw2photonrecycle"          # full photon recycle contact
          fullphotonrecycle = reemit(0.2)
        }
        { Name = "p_Contact"                 # total reflectivity contact
          reflectivity = 1.0
          transmittivity = 0.0
        }
        { Name = "raytrace_pmi"              # raytrace PMI contact
          PMIModel = "rt_pmi_modelname"
        }
    }
}

```

26: Introduction to Lasers and Light-emitting Diodes

Command File Syntax

```
}

# All regions must use ComplexRefractiveIndex model
Physics {
    ComplexRefractiveIndex(
        WavelengthDep ( real imag )
    )
}

# Specify the spectral conversion region
Physics (region="luminescent") {
    SpectralConversion(NumTable)
}

# QWs must be denoted as Active for correct mapping
Physics (region="qw1") {
    MoleFraction(xfraction = 0.15)
    Recombination(-Radiative)
    Active
}
Physics (region="qw2") {
    MoleFraction(xfraction = 0.15)
    Recombination(-Radiative)
    Active
}
}

#####
### Define the electronic solver part here:
#####

Dessis diode {
    Electrode { ... }

    File {
        Grid      = "elec_white_msh.grd"
        Doping   = "elec_white_msh.dat"
        Parameters = "elecparafайл.par"
        Current   = "elsolver"
        Plot      = "elsolver"
        LEDRadiation = "farfield"
        Gain      = "gainfilename"
    }
}

# Choose special LED related plot variables to output
Plot {...}
    LED_TraceSource
    OpticalGeneration
```

```

RayTraceIntensity
RayTrees
}

Physics {...  

    # Need ComplexRefractiveIndex model for raytracer  

    ComplexRefractiveIndex(  

        WavelengthDep ( real imag )  

    )  

    LED (  

        Optics (  

            RayTrace(  

                PolarizationVector = Random           # or (x y z) vector  

                # ----- Set ray starting & termination conditions -----  

                DepthLimit = 10  

                RaysPerVertex = 10  

                minIntensity = 1e-8  

                RaysRandomOffset          # randomize source rays  

#               Wavelength = 888           # [nm] set fixed wavelength  

#               Disable                 # turn off ray tracing  

                # ----- Set output options -----  

                LEDRadiationPara(1000.0,50)      # (<radius-microns>,Npoints)  

                ObservationCenter = (0.0 10.0 500) # (x y z) in microns  

                TraceSource()  

                Print(Skip(5))                  # print every 5th raypath  

                ProgressMarkers = 1             # 1-100% intervals (integer)  

                # ----- Full photon recycling options -----  

                FullPhotonRecycle(  

                    SpectrumEnergySpan = 1.5      # eV  

                    EstimateASEFactor = 1.1       # greater than 1.0  

                    EstimateAbsFactor = 0.3        # between 0.0 and 1.0  

                    PhotonRecycleScaling = 1.0     # scaling for PR power/recom  

                    SpectrumNumberOfPoints = 30    # integer  

                    StartEnergy = 1.63            # eV  

                )  

                PlotEvolvingSpectra( ... )  

                # ----- Spectral conversion options -----  

                ActivateSpectralConversion(  

                    NumberOfPoints = 60          # spectrum discretization  

                    # ----- Phosphor scattering options -----  

#                   ScatterFactor = 0.8         # random scattering  

#                   HenyeyGreensteinScattering(0.9)  

                )  

            )  

        )
    )
}

```

26: Introduction to Lasers and Light-emitting Diodes

Command File Syntax

```
#----- Quantum Well options -----
qwtransport
qwextension = autodetect
qwsatmodel
Strain
qwescattime = 1e-13
qwhscattime = 2e-14
Broadening = 0.04
Lorentzian
)
Thermionic
HeteroInterface
Fermi
}

# Denote QWs as Active regions
Physics (region="qw1") {
    MoleFraction(xfraction=0.4 Grading=0.00)
    Recombination(-Radiative)
    Active
}
Physics (region="qw2") {
    MoleFraction(xfraction=0.4 Grading=0.00)
    Recombination(-Radiative)
    Active
}

# ----- Specify plot parameters -----
GainPlot { range=(1.63,3.1) intervals=70 }
}

#####
### Define the system here:
#####
System {
    diode d1 (p_Contact=vdd n_Contact=gnd) {Physics{OptSolver="opt"}}
    Vsource_pset drive(vdd gnd){ dc = 0.0 }
    Set ( gnd = 0.0 )
    OSolver opt ()
}

#####
### Define solving sequence:
#####
Solve {
    Coupled (Iterations=25) {Poisson}
    Coupled { Poisson Electron Hole Contact Circuit }
    Coupled { Poisson Electron Hole QWhScatter QWeScatter Contact }
```

```

Circuit }

Quasistationary (
    InitialStep = 0.01
    MaxStep = 0.1
    Minstep = 1e-6
    Plot { Range=(0,1) Intervals=5 }
    PlotLEDRadiation { Range=(0,1) Intervals=5 }
    PlotGain { Range=(0,1) Intervals=5 }
    Goal { Parameter=drive.dc Value=3.5 }
)
{
    Plugin (BreakOnFailure) {
        Coupled { Electron Hole Poisson Contact Circuit
                  QWeScatter QWhScatter }
        PhotonRecycle
    }
}

```

Some comments about the command file syntax:

- Computing total radiative recombination rates is performed by numeric integration. To speed up numeric integration, a Gaussian quadrature numeric integration technique is used, and you must specify the activation in the Math section.
 - Multithreading for the raytracer can be activated by specifying Number_Of_Threads in the Math section (see [Monte Carlo Raytracing on page 412](#)). Since raytracing is a recursive function process, the stack space used can overflow the default value. It is recommended to increase the StackSize.
 - Various special raytrace boundary conditions can be chosen and are set in the RayTraceBC section. For details about these special boundary contacts, see [Boundary Condition for Raytracing on page 422](#) and [Defining Photon-Recycling Contacts on page 740](#).
 - The wavelength used in raytracing is computed automatically to be the wavelength at the peak of the spontaneous emission spectrum. If a fixed wavelength is desired, you have the option of setting it by using the Wavelength keyword.
 - The spectral conversion material only needs to be declared in the optical device. There are two choices of phosphor scattering methods: random scattering and Henyey–Greenstein scattering (see [Henyey–Greenstein Volume Scattering Probability on page 753](#)).
 - The QW regions must be labelled as Active in both the optical and electrical device declarations so that proper internal mapping can be performed. You also have to include the keyword Recombination(-Radiative) in the QW active regions because the spontaneous emission computation in these regions uses the special LED model and, therefore, the default radiative recombination model should be switched off.
 - The ComplexRefractiveIndex model must be used in conjunction with the raytracer. In addition, a PolarizationVector must be defined with the raytracer.

26: Introduction to Lasers and Light-emitting Diodes

Command File Syntax

- The FullPhotonRecycle model is recommended for inclusion in all LED simulations because it contains the necessary advanced physical models for LEDs.
- The keyword Print outputs all rays as a grid file. However, if the number of starting rays is large, the output file will be huge, and the high density of rays inside the device will appear as totally black. Additional print options are available (see [Print Options in Raytracing on page 730](#) or [Table 196 on page 1158](#)). Alternatively, the keyword RayTrees can be set in the Plot section to visualize raytracing (see [Visualizing Raytracing on page 429](#)).
- The dual-grid electrical and optical feature has been used. If you select a single-grid simulation, they only need to copy the Physics-LED section of this example and insert it into the Physics section of a typical single-grid command file such as in [Simulating Single-Grid Edge-emitting Laser on page 658](#).
- The photon recycling feature is made self-consistent with the electronic simulation by including the keyword PhotonRecycle in the Plugin loop.

NOTE Raytracing can be disabled in an LED simulation by using the keyword Disable in the RayTrace statement if you do not require the computing of the extraction efficiency and radiation pattern.

[Table 196 on page 1158](#) lists all of the arguments for the LED RayTrace option.

Simulating Organic Light-emitting Diode

The command file syntax for performing a typical organic light-emitting diode (OLED) simulation is:

```
# ----- Define file names for input and output -----
File {
    Grid      = "meshfile.tdr"
    Doping    = "meshfile.tdr"
    Plot      = "dataplotfile.tdr"
    LEDRadiation = "farfieldfile"
    Current   = "currentplotfile.plt"
    Output    = "logfile.log"
    Parameter = "parafiele.par"
}

# ----- Define math options for solvers -----
Math {
    Extrapolate
    RelErrControl
    Notdamped=50
    Iterations=20
    StackSize = 20000000  # increase to 20MB
```

```

}

# ----- Define electrodes -----
Electrode {
    {Name="ITO" Voltage=0.0} # Anode
    {Name="Al" Voltage=0.0} # Cathode
}

# ----- Define the general physics options -----
Physics {
    ComplexRefractiveIndex(
        WavelengthDep( real imag )
    )
    OLED(
        Optics (
            RayTrace(
#            Disable      # run OLED without ray tracing option
            PolarizationVector = Random
            Depthlimit = 1000
            MinIntensity = 1e-3
            LEDRadiationPara(1000.0,50)      # (<radius-microns>,Npoints)
            Symmetry = NonSymmetric
            RaysPerVertex = 20
        )
    )
}
}

# ----- Define the physics of each organic region -----
Physics(Region="HTL-2") {
    Mobility (HighFieldSaturation(PFMob Eparallel))
    EffectiveIntrinsicDensity(NoBandGapNarrowing)
    SingletExciton(Recombination(eQuenching hQuenching Bimolecular Radiative))
    EffectiveMass(GaussianDOS)
    Traps(
        (Acceptor Level Conc=4e17 fromValBand
        EnergyMid=0.25
        eXsection=1e-16
        hXsection=1e-12)
    )
}
Physics(Region="HTL") {
    Mobility (HighFieldSaturation(PFMob Eparallel))
    EffectiveIntrinsicDensity(NoBandGapNarrowing)
    SingletExciton(Recombination(eQuenching hQuenching Bimolecular))
    EffectiveMass(GaussianDOS)
    Traps(
        (Acceptor Level Conc=4e17 fromValBand

```

26: Introduction to Lasers and Light-emitting Diodes

Command File Syntax

```
        EnergyMid=0.25
        eXsection=1e-16
        hXsection=1e-12)
    )
    Active
}
Physics(Region="EBL") {
    Mobility (HighFieldSaturation(PFMob Eparallel))
    EffectiveIntrinsicDensity(NoBandGapNarrowing)
    SingletExciton(Recombination(eQuenching hQuenching Bimolecular Radiative))
    EffectiveMass(GaussianDOS)
    Traps(
        (Acceptor Level Conc=4e17 fromValBand
        EnergyMid=0.25
        eXsection=1e-16
        hXsection=1e-12)
    )
    Active
}
Physics(Region="EML-ETL") {
    Mobility (HighFieldSaturation(PFMob Eparallel))
    EffectiveIntrinsicDensity(NoBandGapNarrowing)
    SingletExciton(Recombination(eQuenching hQuenching Bimolecular Radiative))
    EffectiveMass(GaussianDOS)
    Traps(
        (Donor Level Conc=4e17 fromCondBand
        EnergyMid=0.25
        eXsection=1e-12
        hXsection=1e-16)
    )
    Active
}
Physics(Region="ETL-2") {
    Mobility (HighFieldSaturation(PFMob Eparallel))
    EffectiveIntrinsicDensity(NoBandGapNarrowing)
    SingletExciton(Recombination(eQuenching hQuenching Bimolecular Radiative))
    EffectiveMass(GaussianDOS)
    Traps(
        (Donor Level Conc=4e17 fromCondBand
        EnergyMid=0.25
        eXsection=1e-12
        hXsection=1e-16)
    )
}

# ----- Define organic interface physics -----
Physics(RegionInterface="HTL/EBL") {
    Thermionic(Organic_Gaussian)
```

```
}

Physics(RegionInterface="EBL/EML-ETL") {
    Thermionic(Organic_Gaussian)
}

# ----- Select plot variables to output -----
Plot {
    eCurrent hCurrent Current
    BandGap ConductionBandEnergy ValenceBandEnergy
    ElectronAffinity
    eMobility hMobility
    eRelativeEffectiveMass hRelativeEffectiveMass
    eEffectiveStateDensity hEffectiveStateDensity
    sExc_FluxDensity/Vector
    sExc_EquilibriumDensity
    sExc_eRecomb
    sExc_hRecomb
    sExc_BimolecularRecomb
    eDriftVelocity hDriftVelocity
    SRHRecombination AugerRecombination
    RadiativeRecombination TotalRecombination
    LED_TraceSource
    OpticalGeneration RayTraceIntensity
    RayTrees
}

# ----- Activate solution process -----
Solve {
    Coupled(Iterations=100) {Poisson}
    Coupled {Poisson Electron Hole SingletExciton}

    # ----- Ramp voltage -----
    Quasistationary (
        InitialStep=0.01 Maxstep=0.1 MinStep=1e-4
        PlotLEDRadiation { Range = (0, 1) Intervals = 1 }
        Goal {name="ITO" Voltage=9.0}
    ) {
        Coupled {Poisson Electron Hole SingletExciton}
    }
}
```

26: Introduction to Lasers and Light-emitting Diodes

Investigating Simulation Results

Some comments about the command file syntax:

- In the `File` section, you must specify the list of file names to use for the input and output parameters and results.
- In the `Math` section, it is recommended to increase the `StackSize` because raytracing (see [Raytracing on page 407](#)) is a recursive process and requires a larger stack space (see [Monte Carlo Raytracing on page 412](#)).
- In the general `Physics` section, the keyword `OLED` must be specified. The `OLED` keyword accepts the same range of parameters as the `LED` keyword since the OLED adopts the same code structure as the LED simulator. There is also a `Disable` option that switches off raytracing while allowing the OLED simulator to run, producing only the I–V characteristics and total spontaneous emission power. With the raytracer switched on, you can obtain the far-field radiation pattern and optical intensity (use `RayTraceIntensity` in the `Plot` section) of the device.
- In the `Physics` section of various organic regions, you must define the organic models and parameters that the region contains. For example, in the HTL-2 region, the Poole–Frenkel mobility model (`PFmodel`), the Gaussian DOS (`GaussianDOS`), the singlet excitons, and the traps model are defined.
- The keyword `Active` needs to be included in the `Physics` section of the regions that emit photons (that is, optical emission).
- At organic–organic heterointerfaces, you should activate the organic interface physics model with the keyword `Thermionic(Organic_Gaussian)`.
- In the `Plot` section, you can select many different result variables to plot.
- In the `Solve` section, you should include the `SingletExciton` equation to be coupled to the other transport equations for solving using the Newton method.

Investigating Simulation Results

If the option `Current = "current"` is specified in the `File` section of the command file, Sentaurus Device produces the current file at the end of the simulation. The current file (`current_des plt`) contains laser-specific and LED-specific result variables as a function of bias. These result variables can be plotted using `Inspect`.

Scripts to automatically extract the threshold current, the slope efficiency, and so on are available. You can refer to the TCAD library or contact the Synopsys Technical Support Center to obtain these scripts.

If the option `Plot = "plot"` is specified in the `File` section of the command file, Sentaurus Device creates the plot file at the end of the simulation. The plot file (`plot_des.dat`) contains the plot variables that you have specified in the `Plot` statement, and these plot variables are

given in DF-ISE or TDR format on the vertices of the mesh. The plot variable names are usually meaningful to make it easier to identify what they represent.

Apart from the list of standard plot variables from a Sentaurus Device simulation (see [Appendix F on page 1059](#)), the Laser option introduces an additional set of plot variables.

By inputting the grid file and plot file into Tecplot SV, you can visualize the results.

In addition, there is a list of options that you can switch on to obtain specific output files. These options are discussed in the next sections.

Some options were included in the example in [Simulating Single-Grid Edge-emitting Laser on page 658](#), so that you understand how to activate these options. A summary of these options is:

- Gain curves – Gain versus energy at different biases. If the full photon recycling feature is activated in a LED simulation, the modified spontaneous emission spectrum is also plotted.
- Band structure plots – Results of $k \cdot p$ band structure and wavefunction calculations in quantum well regions
- Far-field plots – Far-field patterns at different biases
- Optical field vector and intensity – Optical intensities at different biases
- VCSEL near field – Transverse VCSEL near field at different biases
- LED radiation – Far zone radiation of an LED emission at different biases and different wavelengths

Current File and Plot Variables for Laser Simulation

[Table 103](#) and [Table 104 on page 683](#) list the current file output and the plot variables for laser simulation.

Table 103 Current file for laser simulation

Dataset group	Dataset	Unit	Description
Time		1 s	For quasistationary. For transient.
n_Contact p_Contact	OuterVoltage	V	
	InnerVoltage	V	
	eCurrent	A	
	hCurrent	A	
	TotalCurrent	A	
	Charge	C	

26: Introduction to Lasers and Light-emitting Diodes

Investigating Simulation Results

Table 103 Current file for laser simulation

Dataset group	Dataset	Unit	Description
TotalPower		W	Total optical output power of all modes.
TotalPhotons		s ⁻¹	Rate of total photon production in the active region of all modes.
Mode0-9	TotalPower	W	Total optical output power of the mode.
	PowerLeft	W	Optical output power at the left (for edge-emitting laser only).
	PowerRight	W	Optical output power at the right (for edge-emitting laser only).
	OpticalGain	cm ⁻¹ s ⁻¹	Model gain at lasing wavelength: Edge-emitting laser VCSEL
	OpticalLoss	cm ⁻¹ s ⁻¹	Model optical loss at lasing wavelength: Edge-emitting laser VCSEL
	TotalOutcouplingLoss	cm ⁻¹ s ⁻¹	Model optical outcoupling loss (edge-emitting laser). Decay rate through top mirror (VCSEL).
	OutcouplingLossLeft	cm ⁻¹	For edge-emitting laser only.
	OutcouplingLossRight	cm ⁻¹	For edge-emitting laser only.
	SpontEmission	cm ⁻¹ s ⁻¹	Model spontaneous emission: Edge-emitting laser VCSEL
	Wavelength	nm	Lasing wavelength.
	RefIndexChange	1	
	EffectiveIndex	1	
	OpticalConfinementFactor	1	

Table 104 Plot variables for laser simulation

Plot variable	Dataset name	Unit	Description
LaserIntensity	OpticalIntensity	Wm^{-3}	Total laser intensity for multimode lasing.
DielectricConstant		1	Dielectric profile.
RefractiveIndex		1	Refractive index profile.
MatGain	OpticalMaterialGain	m^{-1}	Local material gain.
OpticalIntensityMode0..9		Wm^{-3}	Optical intensity of mode 0 to mode 9.
OpticalPolarizationAngleMode0..9		rad	Polarization angle of the optical field vector (0 for TE, $\pi/2$ for TM).
eDifferentialGain hDifferentialGain		cm^2	$\partial r^{\text{st}} / \partial n, \partial r^{\text{st}} / \partial p$ fundamental mode only.
StimulatedRecombination SpontaneousRecombination		$\text{cm}^{-3} \text{s}^{-1}$	Sum of stimulated and spontaneous emission in all modes.

Current File and Plot Variables for LED Simulation

[Table 105](#) and [Table 106 on page 684](#) list the current file output and the plot variables valid for LED simulation.

Table 105 Current file for LED simulation

Dataset group	Dataset	Unit	Description
Time		1 s	For quasistationary. For transient.
n_Contact p_Contact	OuterVoltage	V	
	InnerVoltage	V	
	eCurrent	A	
	hCurrent	A	
	TotalCurrent	A	
	Charge	C	
LedWavelength		nm	Average wavelength of LED simulation.
Photon_ExtEfficiency		1	Photon extraction efficiency.

26: Introduction to Lasers and Light-emitting Diodes

Investigating Simulation Results

Table 105 Current file for LED simulation

Dataset group	Dataset	Unit	Description
Photon_Spontaneous		s ⁻¹	Spontaneous emission photon rate.
Photon_Exited		s ⁻¹	Rate of photon escaping from device.
Photon_NetPhotonReclycle		s ⁻¹	Net photon-recycling photon rate.
Photon_NonActiveAbsorb		s ⁻¹	Rate of photon absorption in nonactive regions.
Power_ExtEfficiency		1	Power extraction efficiency.
Power_Spontaneous		W	Spontaneous emission power.
Power_ASE		W	Amplified spontaneous emission power.
Power_ReEmit		W	Re-emission power.
Power_Absorption		W	Absorption power.
Power_SpecConvertGain		W	Net power gain of spectral conversion.
Power_NetPhotonRecycle		W	Net photon-recycling power.
Power_Total		W	Total internal optical power of LED.

Table 106 Plot variables for LED simulation

Plot variable	Dataset name	Unit	Description
DielectricConstant		1	Dielectric profile.
RefractiveIndex		1	Refractive index profile.
MatGain	OpticalMaterialGain	m ⁻¹	Local material gain.
LED_TraceSource			Influence of each active vertex on the total extracted light.
SpontaneousRecombination		cm ⁻³ s ⁻¹	Sum of spontaneous emission.
RayTraceIntensity		Wcm ⁻³	Optical intensity from raytracing.
RayTrees			Raytree structure.

`LedWavelength` is computed automatically in the simulation. It is taken as the wavelength where the peak of the spontaneous spectrum occurs. For clarity, photon rate and power output results are separated into two groups:

- The photon rate group has units of s^{-1} .
- The power group has units of W.

Photon and power quantities need to be computed separately if a spectrum is involved. Suppose that the spontaneous emission coefficient is r^{sp} (units of $eV^{-1} cm^{-3} s^{-1}$). The photon rate is $\int r^{sp} dE$ while the power is $\int r^{sp} \cdot E dE$. The extraction coefficient is the ratio of the excited and internal quantities, so the photon rate (`Photon_ExtEfficiency`) and power (`Power_ExtEfficiency`) extraction efficiencies are different.

The only case when `Photon_ExtEfficiency=Power_ExtEfficiency` is for a single wavelength simulation that does not involve a spectrum.

The total outcoupled (excited) power is then (`Power_ExtEfficiency * Power_Total`), where the total internal optical power is:

$$\text{Power_Total} = \text{Power_Spontaneous} + \text{Power_NetPhotonRecycle} + \text{Power_SpecConvertGain} \quad (710)$$

The photon rate extraction efficiency has been computed using:

$$\text{Photon_ExtEfficiency} = \text{Photon_Exited} / (\text{Photon_Spontaneous} + \text{Photon_NetPhotonRecycle}) \quad (711)$$

Plot Variables for Quantum-Well Modeling

[Table 107](#) lists the plot variables valid for quantum-well (QW) modeling.

Table 107 Plot variables for QW modeling

Plot variable	Dataset name	Unit	Description
<code>QWeDensity</code> <code>QWhDensity</code>	<code>QW_eCarrDensity</code> <code>QW_hCarrDensity</code>	cm^{-3}	QW electron and hole bound-state densities.
<code>QWeQuasiFermi</code> <code>QWhQuasiFermi</code>	<code>QW_eQuasiFermi</code> <code>QW_hQuasiFermi</code>	eV	QW electron and hole quasi-Fermi levels.
<code>QW_eNetCapture</code> <code>QW_hNetCapture</code>		$cm^{-3} s^{-1}$	Net QW capture rates of electrons and holes.
<code>QW_eLeakageCurrent</code> <code>QW_hLeakageCurrent</code>		Acm^{-2}	Electron and hole leakage current density across the QW.

26: Introduction to Lasers and Light-emitting Diodes

Investigating Simulation Results

This chapter describes the physics and the models used in laser simulations.

For laser simulations, Fabry–Perot cavity and VCSEL cavity treatments are discussed. Small-signal photon intensity and phase modulation, and relative intensity and phase noise are also described.

Overview

Simulating a laser diode is one of the most complex problems in device simulation. The fundamental set of equations used in a laser simulation is:

- Poisson equation
- Carrier continuity equations
- Lattice temperature equation and hydrodynamic equations
- Quantum well scattering equations (for QW carrier capture)
- Quantum well gain calculations (Schrödinger equation)
- Photon rate equation
- Helmholtz equation

The first three equations are semiconductor transport equations for the drift-diffusion regime (see [Transport Equations on page 177](#)). Other than the semiconductor transport equations, which solve the electrical drift-diffusion problem for holes and electrons, the carrier capture in the quantum well (QW) and the gain calculations are specific to the laser problem, and they link the electronics and optics. The QW carrier capture and gain calculations are discussed in [Chapter 30 on page 803](#). The solution of the Helmholtz equation provides the optical modes and other optical quantities, and this is presented in [Chapter 29 on page 761](#). Finally, the ‘moderator’ between the electronics and optics is the photon rate equation, which solves for the total number of photons. In the next section, the relationship between all these equations is shown and how to achieve self-consistency between all these equations is explained.

Coupling Between Optics and Electronics

The coupling between the different equations in the laser simulation is illustrated in Figure 66.

The complexity of the laser problem is apparent from this relational chart. The key quantities exchanged between different equations are placed alongside the directional flows between the equation blocks. The optical problem is separated from the electrical problem. For the optical problem, an eigenvalue equation is solved, and the eigenvector (mode intensity) and eigenvalue (effective index or mode frequency) are inserted into the electrical equations and the photon rate equation. As a result, the coupling between the optics and electronics becomes nonlocal, making it difficult to solve these problems simultaneously by the Newton method. Therefore, a Gummel iteration method (instead of a coupled Newton iteration) is required to couple the electrical and optical problems self-consistently.

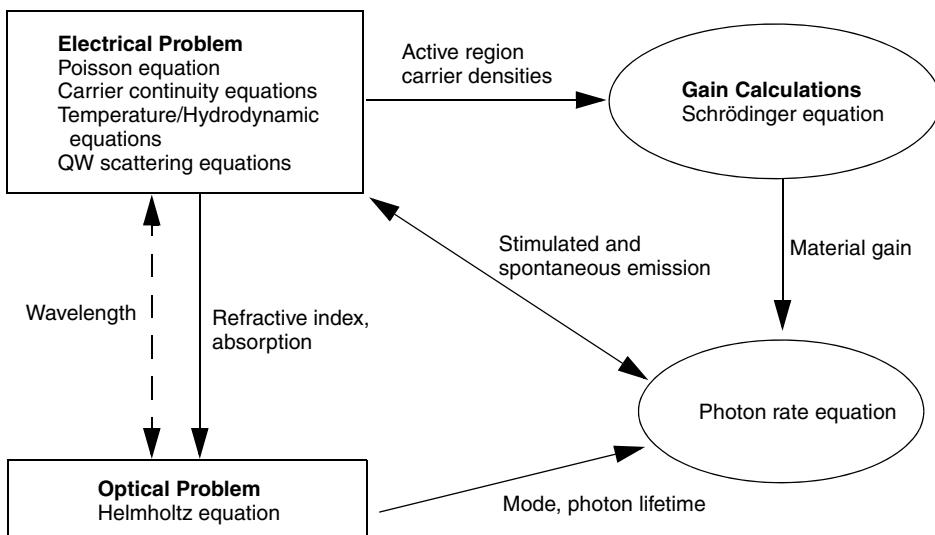


Figure 66 Coupling between the semiconductor transport and optics equations

The solution of the electrical problem provides the refractive index and absorption to the optical solver, which solves for the modes. In the case of the Fabry–Perot edge-emitting laser, the wavelength is computed from the peak of the gain curve and fed into the optical problem. In VCSELs and distributed Bragg reflector (DBR) lasers, the wavelength is computed as part of the optical resonance problem and is an input to the electrical problem instead.

The gain calculations involve the solution of the Schrödinger equation for the subband energy levels and wavefunctions. These quantities are used with the active-region carrier statistics to compute the optical matrix elements for stimulated and spontaneous emission. The formulation of these emission processes is based on Fermi's golden rule. For more details about the gain calculations, see [Chapter 30 on page 803](#).

The optical recombination processes increase the photon population, thereby reducing the carrier population. Therefore, they are included in the photon rate equation that is used to calculate the number of photons present in the laser device and also in the carrier continuity equations to ensure the conservation of particles (see [Photon Rate and Photon Phase Equations on page 690](#)).

Algorithm for Coupling Electrical and Optical Problems

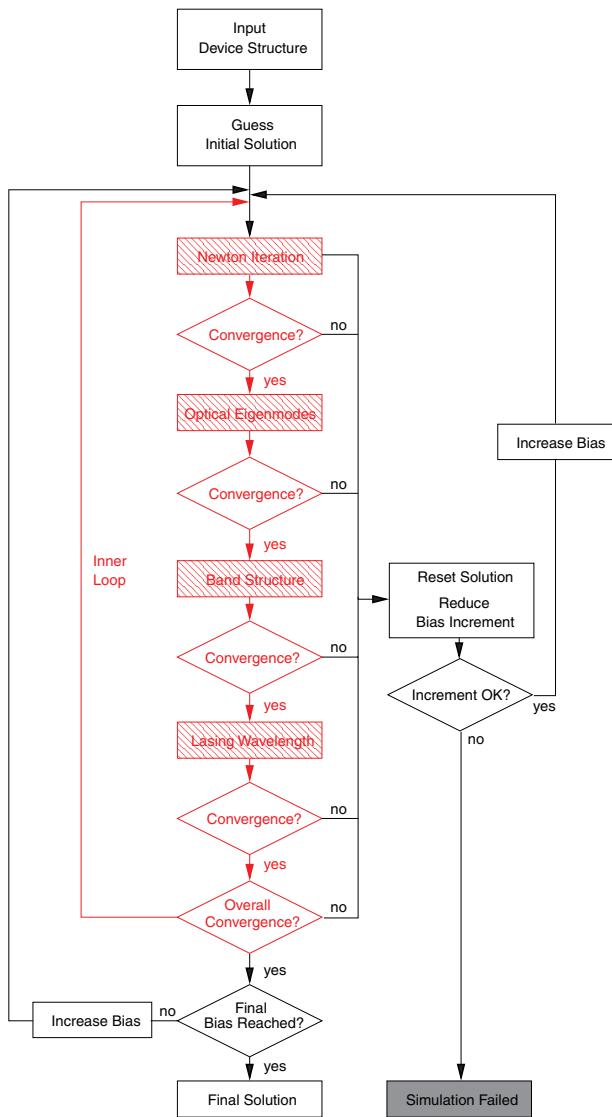


Figure 67 Algorithm flowchart for the self-consistent solution of laser equations

27: Lasers

Photon Rate and Photon Phase Equations

The Newton iteration is performed in Sentaurus Device using the `Coupled` keyword, while the Gummel iteration is activated using the keyword `Plugin`. [Figure 67 on page 689](#) shows the algorithm flowchart for the laser simulation. In addition, refer to the `Solve` section in [Simulating Single-Grid Edge-emitting Laser on page 658](#) for details about the syntax.

The statement `Coupled {Electron Hole Poisson QWeScatter QWhScatter PhotonRate}` activates the Newton iterations for the electron and hole continuity equations, the Poisson equation, the QW carrier capture equations, and the photon rate equation. This is indicated by the Newton iteration block in [Figure 67](#).

After the convergence of the Newton iterations, the updated refractive index profile (if it changes with temperature and carrier densities) is passed to the optical solver to solve for the optical eigenmodes. At this instance, the band structures are also computed. In the case of VCSELs and DBR lasers, the resonant (or lasing) wavelength is also computed by the optical solver. The keyword `Plugin` ensures that the electrical and optical solutions are iterated self-consistently, that is, a Gummel-type iteration. Through this self-consistency, all the physics should be satisfied. Sentaurus Device automatically handles the flow of the result variables between the electrical and optical solvers. There are also additional checkpoints to restrict the solution of each part from diverging during the Gummel iteration, as shown in [Figure 67](#).

When convergence is attained for the Gummel iteration (`Plugin`), the solution set for this bias is used as the initial guess for the next bias. In this way, the continuous wave operation of the laser diodes can be simulated.

Photon Rate and Photon Phase Equations

The origin of the photon rate equation and the photon phase equation can be traced to the famous work of Henry [1] on the theory of spontaneous emission noise in open resonators. The cavity of an edge-emitting Fabry–Perot laser is considered an open electromagnetic resonator, and the modes are driven by the radiative recombination processes.

From the Maxwell equations, the wave equation for the electric field is derived as:

$$\nabla \times \nabla \times \mathbf{E} = -\mu_0 \left(\sigma \frac{\partial \mathbf{E}}{\partial t} + \epsilon_0 \epsilon_r \frac{d^2 \mathbf{E}}{dt^2} + \frac{d^2 \mathbf{P}}{dt^2} \right) \quad (712)$$

where \mathbf{E} is the electric field, σ is the conductivity, and ϵ_r is the relative permittivity. The source term \mathbf{P} can be identified as the polarization due to the spontaneous recombination of electron–hole pairs. For the electric field, the following separation ansatz is made:

$$\mathbf{E}(x, y, z, t) = \Psi(x, y; t) \sqrt{S(t)} e^{i\phi(t)} e^{i(\omega_0 t - \kappa_0 z)} + cc \quad (713)$$

where cc denotes the complex conjugate. The optical field, Ψ , is complex; while the photon density, $S(t)$, and phase factor, φ , are real quantities. Ψ is normalized according to:

$$\iint |\Psi|^2 dx dy = 1 \quad (714)$$

Inserting Eq. 713 into Eq. 712 results in three equations:

(i) The Helmholtz equation for $\Psi(x, y)$, which is discussed in [Chapter 30 on page 803](#):

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) \Psi + k_0^2(n^2(x, y) - \epsilon_{eff}) \Psi = 0 \quad (715)$$

(ii) The equation for the photon phase $\Phi(t)$:

$$\frac{\partial \Phi}{\partial t} - [G(\omega) - L] \frac{\left(\frac{c}{n}\right) \alpha_v}{2} = 0 \quad (716)$$

where $G(\omega)$ is the modal gain, L is the optical loss, and the line-width enhancement factor α_v must be entered by you (using the keywords `PhotonPhaseAlpha` and `LineWidthEnhancementFactor`).

(iii) The photon rate equation for the photon number $S(t)$:

$$\frac{\partial S}{\partial t} - (G(\omega) - L)S = \frac{c}{\epsilon_r} \beta T^{sp}(\omega) \quad (717)$$

The photon rate equation contains the modal spontaneous emission, $T^{sp}(\omega)$, and the spontaneous emission factor β that can be defined by you (using the keyword `SponEmiss`) and that has a default value of 1.

The parameters are defined as:

$$G(\omega) = \iint r^{st}(x, y, E_{\omega}) |\Psi(x, y)|^2 dx dy \quad (718)$$

$$T^{sp}(\omega) = \iint r^{sp}(x, y, E_{\omega}) |\Psi(x, y)|^2 dx dy \quad (719)$$

$$L = \left(\frac{c}{n} \right) \left\{ \iint \alpha(x, y) |\Psi(x, y)|^2 dx dy + L_{bg} + L_{cavity} + L_{waveguide} \right\} \quad (720)$$

27: Lasers

Photon Rate and Photon Phase Equations

The stimulated emission coefficient $r^{\text{st}}(x, y, E_{\omega})$ and the spontaneous emission coefficient $r^{\text{sp}}(x, y, E_{\omega})$ are computed locally at each active vertex from Fermi's golden rule, and their values are taken at the lasing energy E_{ω} . These radiative emission coefficients are discussed in [Chapter 30 on page 803](#). The local optical intensity $|\Psi(x, y)|^2$ is solved from the Helmholtz equation. In the simulation, the spatial integrations are performed in the active regions of the laser only.

The total optical loss L contains the loss due to local free carrier absorption $\alpha(x, y)$, the cavity loss L_{cavity} , the background optical loss L_{bg} , and the waveguide loss $L_{\text{waveguide}}$. The free carrier loss is described in detail in [Free Carrier Loss on page 702](#). You specify the background loss. The waveguide loss (imaginary part of propagation constant) is fed back from the optical solver to the photon rate equation, and this includes the losses due to leakage waves from the waveguide.

The cavity loss for a Fabry–Perot cavity mainly contains the mirror loss and is given by:

$$L_{\text{cavity}} = \frac{1}{2l} \ln\left(\frac{1}{r_0 r_1}\right) \cdot \frac{c}{n} \quad (721)$$

where l is the cavity length, r_0 and r_1 are the facet power reflectivities, c is the speed of light in a vacuum, and n is modal effective refractive index.

This formula is used mainly in the simulation of edge-emitting lasers. For VCSELs, the cavity loss is more complicated due to scattering and diffraction effects, and can only be accurately computed by a vectorial cavity solution. In this case, the net loss in the VCSEL photon rate equation, $(L - G(\omega))$, must be solved from the optics equation when gain-guiding effects are included. This is discussed more fully in [Cavity Optical Modes in VCSELs on page 711](#).

Each optical mode corresponds to a distinct photon rate equation. For example, if you specify up to ten modes in a laser simulation, Sentaurus Device automatically solves up to ten photon rate equations. The stimulated and spontaneous emissions from each photon rate equation are then summed to give a total radiative emission rate. This total rate is then added to the carrier continuity equation as a radiative recombination to ensure the conservation of electron–hole pair recombination and photon emission.

At lasing threshold, the modal gain of the laser saturates to the value of the optical losses, which results in a singularity in the steady-state photon rate:

$$S = \left(\frac{\bar{\epsilon}_r}{c}\right) \cdot \frac{\beta T^{\text{sp}}(\omega)}{L - G(\omega)} \quad (722)$$

A small fluctuation of the QW carrier densities and, therefore, gain during Newton iterations leads to large changes in the photon rate. In cases where the gain exceeds the loss, the photon rate suddenly becomes negative, which is nonphysical and pushes the entire Newton iteration towards divergence. To solve this problem, a slack variable, λ_{slack} , is introduced to constrain $(L - G(\omega))$ to be always positive:

$$\lambda_{\text{slack}}^2 = L - G(\omega) \quad (723)$$

so that the photon rate will always be positive. This important numeric technique to laser simulation was first proposed by Smith *et al.* [2] and, in Sentaurus Device, it is called the photon stabilization equation. When the photon rate equation is activated, Sentaurus Device automatically activates the photon stabilization equation as well.

The syntax for specifying the various parameters in the photon rate and photon phase equations can be found in [Table 133 on page 1100](#) and [Table 182 on page 1150](#).

Laser Small-Signal AC Analysis and Modulation Response

Small-signal AC analysis for the laser intensity and photon phase is analogous to the small-signal AC analysis described in [ACCoupled: Small-Signal AC Analysis on page 161](#).

Intensity and Photon Phase Modulation Response

The dynamic laser device properties under current modulation can be calculated from a converged static solution of the system of [Eq. 26, p. 178](#), [Eq. 27, p. 178](#), [Eq. 32, p. 180](#), [Eq. 716](#), and [Eq. 729](#) using a small-signal expansion. The intensity modulation response can then be described as the change in the modal photon number S_v of a mode v caused by a change in the injection current I :

$$\partial S_v(\omega) = \iiint G_{\partial I}^{S_v}(r, \omega) \partial I(r, \omega) dV \quad (724)$$

where $G_{\partial I}^{S_v}$ is the Green's function of the photon number S_v due to a current excitation. Analogously, the modulation response of the photon phase is given by:

$$\partial \Phi_v(\omega) = \iiint G_{\partial I}^{\Phi_v}(r, \omega) \partial I(r, \omega) dV \quad (725)$$

The Green's functions are obtained by inverting the Fourier-transformed Jacobian matrix of the converged electro-optothermal system of [Eq. 26](#), [Eq. 27](#), [Eq. 32](#), [Eq. 716](#), and [Eq. 729](#).

Syntax for Small-Signal AC Extraction

The syntax for small-signal AC analysis for lasers is analogous to the syntax for other devices as described in [ACCoupled: Small-Signal AC Analysis on page 161](#). First, the device must be ramped up to its operating point, before the AC analysis can be performed:

```

Solve {...}
    # Ramp up to operating point:
    Quasistationary (
        InitialStep = 0.01
        MaxStep = 0.05
        MinStep = 1e-5
        Goal { Parameter=i_drive.dc value = -5e-3 } )
        { Coupled{ Poisson Hole Electron Contact Circuit qwhScatter qweScatter
            PhotonRate }
    }

    # Perform chirp analysis:
    ACCoupled (
        StartFrequency = 1e8 EndFrequency = 1e12
        ACEExtract = "ac_chirp"
        NumberOfPoints = 100
        Decade
        Node(nanode)
        Exclude(i_drive)
        PhotonicAC)
        { Poisson Hole Electron Contact Circuit qwhScatter qweScatter PhotonRate
            PhotonPhase }
    )
}

```

For a detailed description of the arguments of the ACCoupled statement, see [ACCoupled: Small-Signal AC Analysis on page 161](#) and [Table 129 on page 1097](#).

NOTE A small-signal analysis can also be performed for the calculation of the relative intensity noise and the frequency noise of a laser (see [Modeling Relative Intensity Noise and Frequency Noise on page 695](#)).

Modeling Relative Intensity Noise and Frequency Noise

Sentaurus Device contains a spatially distributed noise model for the simulation of relative intensity noise (RIN) and frequency noise (FN) in semiconductor lasers. The model is based on spatially distributed Langevin forces and correlation functions that are formulated in the frequency domain assuming small-signal noise sources.

Relative Intensity Noise and Frequency Noise

To model noise in lasers, the continuity equations (Eq. 27, p. 178) and the photon phase and photon rate equations (Eq. 716 and Eq. 717) are extended by appropriate local Langevin noise forces $F_n(t)$, $F_p(t)$, $F_\Phi(t)$, and $F_S(t)$ for electrons, holes, photon phase, and photon rate, respectively:

$$\nabla \cdot \vec{J}_n = qR + q\frac{\partial n}{\partial t} + F_n(t) \quad -\nabla \cdot \vec{J}_p = qR + q\frac{\partial p}{\partial t} + F_p(t) \quad (726)$$

$$\frac{\partial \Phi}{\partial t} - (G(\omega) - L)\frac{\left(\frac{c}{n}\right)\alpha_v}{2} = F_\Phi(t) \quad (727)$$

$$\frac{\partial S}{\partial t} - (G(\omega) - L)S = \frac{c}{\epsilon_r} \beta T^{\text{sp}}(\omega) + F_S(t) \quad (728)$$

Relative intensity noise in a laser is defined as a function of frequency f according to:

$$\frac{\text{RIN}(f)}{\Delta f} = \text{Re} \left(\frac{\sum_{\mu, v} S_{S_v S_\mu}(f)}{\sum_v S_v^2} \right) \quad (729)$$

where $S_{S_v S_\mu}$ denotes the correlation function between the photon numbers S_v and S_μ of modes v and μ .

The line width of mode v is defined as:

$$\Delta f = \lim_{f \rightarrow 0} S_{S_v S_v}(f) \quad (730)$$

where $S_{S_v S_v}$ is the autocorrelation function of the lasing frequency of mode v , which can be interpreted as frequency noise.

27: Lasers

Modeling Relative Intensity Noise and Frequency Noise

This is related to the photon phase autocorrelation function $S_{\Phi_v \Phi_v}$ by:

$$S_{S_v S_v}(f) = f^2 S_{\Phi_v \Phi_v}(f) \quad (731)$$

Therefore, RIN and line width can be calculated through the correlation functions of the photon number and photon phase by either explicit calculations in the time domain or assuming small-signal noise sources. As the time domain approach is computationally very expensive, Sentaurus Device uses the small-signal approach.

Small-Signal Noise Modeling

For the computation of RIN, the correlation functions of the modal photon numbers $S_{S_v S_\mu}$ must be evaluated:

$$S_{S_v S_\mu}(\omega) = \sum_{\gamma} \sum_{\delta} \int \int \int \hat{G}_\gamma^{S_v}(r_1, \omega) K_{\gamma, \delta}(r_1, \omega) \hat{G}_\delta^{S_v*}(r_1, \omega) dr_1 \quad (732)$$

where the summations are over the other system variables (potential, densities, photon number, and photon phase). $\hat{G}_\gamma^{S_v}(r_1; \omega)$ is the Green's function of the photon number S_v caused by an excitation with quantity γ at a coordinate r_1 with an angular frequency ω and $*$ denotes the complex conjugate.

$K_{\gamma, \delta}(r_1, \omega)$ denotes the noise source correlation between the quantities γ and δ at r_1 . The Green's function can be obtained from the Fourier-transformed Jacobian matrix of the system of [Eq. 26](#), [Eq. 32](#), [Eq. 726](#), [Eq. 727](#), and [Eq. 728](#). According to [Eq. 732](#), frequency noise for mode v can be calculated using:

$$S_{\Phi_v \Phi_\mu}(\omega) = \sum_{\gamma} \sum_{\delta} \int \int \int \hat{G}_\gamma^{\Phi_v}(r_1, \omega) K_{\gamma, \delta}(r_1, \omega) \hat{G}_\delta^{\Phi_v*}(r_1, \omega) dr_1 \quad (733)$$

The local noise sources $K_{\gamma, \delta}(r_1, \omega)$ are given by:

$$K_{\gamma, \delta} = \int \langle F_\gamma(t) F_\delta^*(t - \tau) \rangle \exp(-i\omega\tau) dt \quad (734)$$

where the angle brackets denote the expectation value.

Sentaurus Device includes only the autocorrelation functions for the photon numbers and the photon phases. The terms for the $K_{\gamma, \delta}(r_1, \omega)$ are given by:

$$K_{S_v, S_v} = 2T_v^{sp} S_v \quad K_{\Phi_v, \Phi_v} = \frac{T_v^{sp}}{2S_v} \quad (735)$$

Syntax for Relative Intensity Noise and Frequency Noise Model

The noise model described here can be used with the small-signal AC analysis described in [Laser Small-Signal AC Analysis and Modulation Response on page 693](#) to obtain the frequency-resolved relative intensity noise (RIN) and frequency noise (FN) of a laser device.

The syntax to activate the noise sources described in [Eq. 735](#) is:

```
Physics {
    Noise(PhotonNumberNoise PhotonPhaseNoise)
}
```

where `PhotonNumberNoise` and `PhotonPhaseNoise` denote K_{S_v, S_v} and K_{Φ_v, Φ_v} , respectively.

The syntax for `ACCoupled` differs to the syntax described in [Syntax for Small-Signal AC Extraction on page 694](#) only in the missing `Node` keyword, as there are no terminals for which the analysis is performed.

The syntax for RIN is:

```
ACCoupled(
    StartFrequency = 1e8 EndFrequency = 1e12
    ACEExtract = "int_noise"
    NumberOfPoints = 100
    Decade
    Exclude(i_drive)
    PhotonicIntensityNoise)
{ Poisson Hole Electron Contact Circuit qwhScatter qweScatter PhotonRate
    PhotonPhase }
```

For frequency noise, the syntax is:

```
ACCoupled(
    StartFrequency = 1e8 EndFrequency = 1e12
    ACEExtract = "int_noise"
    NumberOfPoints = 100
    Decade
    Exclude(i_drive)
    PhotonicPhaseNoise)
{ Poisson Hole Electron Contact Circuit qwhScatter qweScatter PhotonRate
    PhotonPhase }
```

27: Lasers

Plotting the Laser Power Spectrum

For more information about the syntax of the ACCoupled statement, see [ACCoupled: Small-Signal AC Analysis on page 161](#) and [Table 129 on page 1097](#).

Plotting the Laser Power Spectrum

The power spectrum of a semiconductor laser can be calculated to a good approximation according to:

$$P_{\text{out}}(\hbar\omega) = \sum_v \hbar\omega_v \cdot L_{v,\text{out}} \cdot S_v \cdot \frac{(\Delta\omega_v/2)^2}{(\omega - \omega_v)^2 + (\Delta\omega_v/2)^2} \quad (736)$$

where ω_v is the lasing frequency, $\hbar\omega_v$ is the lasing energy of mode v , $L_{v,\text{out}}$ is the power out-coupling factor of the mode, and S_v is the number of photons in the mode. $\Delta\omega_v$ is the FWHM Lorentzian broadening of mode v , which is given by the expression:

$$\Delta\omega_v = \frac{(1 + \alpha_H)^2}{2S_v} \cdot R_v^{\text{sp}} \quad (737)$$

with the line width enhancement factor α_H and the total spontaneous emission rate R_v^{sp} into mode v . As power spectrum plots are usually used to investigate detuning effects, the plotting of such spectra is coupled to the plotting of modal gain spectra. The plotting of laser spectra is activated by adding the keyword PlotLasingSpectrum to the GainPlot section of the command file. The file names of power spectra plots are derived from the base name for gain plots:

```
File {...  
    Gain = "gn"  
}  
GainPlot{ Range=(1.22,1.32)      # energy range [eV]  
         Intervals=120          # discretization of energy range  
         PlotLasingSpectrum } # also plot power spectrum  
...  
Solve {...  
    Quasistationary (...  
        # ----- Specify plot gain parameters -----  
        PlotGain {Range=(0,1) Intervals=3}  
        ...  
    {...}  
}
```

This syntax example results in the modal gain and spontaneous emission being plotted into the files `gn_gain_000000_des.plt`, ..., `gn_gain_000003_des.plt`, as well as the laser power spectra being plotted into `gn_powspec_000000_des.plt`, ...,

`gn_powspec_000003_des.plt`. For details about the syntax of this example, see [Modal Gain as Function of Energy/Wavelength on page 856](#).

Refractive Index, Dispersion, and Optical Loss

The refractive index (dielectric constant) is a function of temperature, carrier density, and wavelength. The temperature dependence is assumed to be linear and the change in refractive index due to carriers is attributed to the free carrier plasma effect [3].

You can specify the refractive index dependence on temperature and carrier density with the keywords `TemperatureDep` and `CarrierDep` in the `Physics-Laser` section of the command file:

```
Physics {...  
  Laser (...  
    RefractiveIndex(TemperatureDep CarrierDep)  
  )  
}
```

You can select either or both type of dependence. If none is chosen (that is, there are no keywords), Sentaurus Device assumes that the refractive index is a constant value.

NOTE The refractive index and its associated temperature parameters for each material region can be changed by you in the parameter file.

Temperature Dependence of Refractive Index

The temperature dependence of the refractive index follows the relation:

$$n(T) = n \cdot (1 + \alpha(T - T_{\text{par}})) \quad (738)$$

and the coefficients can be changed in the parameter file:

```
RefractiveIndex  
{ * Optical Refractive Index  
  * refractiveindex() = refractiveindex * (1 + alpha * (T-Tpar))  
    Tpar = 3.0000e+02      # [K]  
    refractiveindex = 3.60e+00  
    alpha = 0.0000e-04      # [1/K]  
}
```

Carrier-Density Dependence of Refractive Index

The change in refractive index due to free carriers [3] is:

$$\Delta n = -\text{CarrDepCoeff} \times \frac{e^2 \lambda^2}{8\pi^2 c^2 \epsilon_0 n_g} \left(\frac{n}{m_e} + \frac{p}{m_h} \right) \quad (739)$$

where `CarrDepCoeff` is introduced as a tuning parameter and has a default value of 1. λ is the lasing wavelength and n_g is the refractive index. n_g can also change if it is specified as `TemperatureDep`.

In the case of QW carriers, the heavy-hole mass is modified to become [3]:

$$m_h = \frac{m_{hh}^{1.5} + m_{lh}^{1.5}}{\sqrt{m_{hh}} + \sqrt{m_{lh}}} \quad (740)$$

The activating syntax in the command file is:

```
Physics {...  
  Laser (...  
    Optics (...)  
      RefractiveIndex(CarrierDep)          # use lasing wavelength of mode0  
    #   RefractiveIndex(CarrierDep(LasingWavelength=777.77))  
          # fixed wavelength [nm]  
  )  
}
```

There is an option to fix a `LasingWavelength` if required. Otherwise, Sentaurus Device automatically uses the lasing wavelength that it computes in the simulation. The tuning parameter `CarrDepCoeff` is defined for each region and can be input in the parameter file:

```
RefractiveIndex  
{ * Optical Refractive Index  
  refractiveindex = 3.893 # [1]  
  * Mole fraction dependent model.  
  * If just above parameters are specified, then its values will be  
  * used for any mole fraction instead of an interpolation below.  
  * The linear interpolation is used on interval [0,1].  
  refractiveindex(1) = 3.51      # [1]  
  
  * Tune the region-wise carrier dependence (plasma effect)  
  *           e^2.lambda^2      ( n      p )  
  * del_n = - CarrDepCoeff * ----- ( --- + --- )  
  *           8pi^2.c^2.epsilon0.n_refr ( m_e    m_h )  
  * Default of CarrDepCoeff is 1.
```

```

CarrDepCoeff = 1.0      # [1]
CarrDepCoeff(0) = 0.5
CarrDepCoeff(1) = 1.0
}

```

If `CarrDepCoeff` is not entered in the parameter file, it is assumed to be the default value of 1 during the simulation. `CarrDepCoeff` is a mole fraction-dependent parameter, so it works the same way as the other mole fraction-dependent parameters.

Wavelength Dependence and Absorption of Refractive Index

Wavelength-dependent optical absorption can also be included by using the `TableODB` section in the parameter file:

```

Physics {...}
  Laser (...)
    Optics (...)
      FEVectorial (...)
        Absorption(ODB)
      )
    )
  )
}

```

This feature is also available for the other optical solvers such as `FEScalar`, `TMM1D`, and `EffectiveIndex`.

In this case, you must create a corresponding ODB table for each material region in the parameter file, for example:

```

TableODB
{ * WAVELEN(um) n k
  0.5904 3.940 0.240
  0.6199 3.878 0.211
  0.6526 3.826 0.179
  0.6888 3.785 0.151
  0.7293 3.742 0.112
  0.7749 3.700 0.091
  0.8266 3.666 0.080
  0.8856 3.614 0.0017
  0.9184 3.569 0.0
  1.0332 3.492 0.0
  1.1271 3.455 0.0
  1.2399 3.423 0.0
  1.3776 3.397 0.0
}

```

27: Lasers

Free Carrier Loss

```
1.5498 3.374 0.0
1.7712 3.354 0.0
2.0664 3.338 0.0
2.4797 3.324 0.0
}
```

Free Carrier Loss

The free carrier loss is caused by plasma-induced effects and contributes to the total optical loss as shown in [Eq. 720](#). The free carrier loss can be modeled by:

$$L_{\text{carr}} = \iint (\alpha_n n + \alpha_p p) |\Psi|^2 dV \quad (741)$$

where the loss is linearly dependent on the electron and hole carrier densities. This model is activated by the keyword `FreeCarr` in the `Physics-Laser` section of the command file:

```
Plot {...  
    eFreeCarrierAbsorption  
    hFreeCarrierAbsorption  
}  
...  
Physics {...  
    Laser (...  
        Optics (...  
            FEVectorial (...  
                )  
        )  
        FreeCarr  
    )  
}
```

In this example, `eFreeCarrierAbsorption` and `hFreeCarrierAbsorption` in the `Plot` section enable the electron and hole contributions to the free carrier loss to be plotted. The coefficients for the free carrier loss, α_n and α_p , for each material region are specified in the parameter file:

```
FreeCarrierAbsorption  
{  
    * Coefficients for free carrier absorption:  
    * alpha_n for electrons,  
    * alpha_p for holes  
  
    * FCA = (alpha_n * n + alpha_p * p) * Light Intensity  
    * Mole fraction dependent model.  
    * If only constant parameters are specified, those values will be  
    * used for any mole fraction instead of the interpolation below.
```

```
* Linear interpolation is used on the interval [0,1].
fcaalpha_n(0) = 4.0000e-18    # [cm^2]
fcaalpha_n(1) = 5.0000e-18    # [cm^2]
fcaalpha_p(0) = 8.0000e-18    # [cm^2]
fcaalpha_p(1) = 9.0000e-18    # [cm^2]
}
```

Edge-emitting Lasers

The 2D edge-emitting laser structure is essentially a waveguide problem (see [Waveguide Optical Modes and Fabry–Perot Cavity](#)) in the transverse plane, with the cavity resonance occurring in the longitudinal direction. The main type of cavity resonance in the longitudinal direction used is the Fabry–Perot type where the resonant wavelength is chosen as the location of the gain peak. There is also an option to activate a simple distributed feedback (DFB) laser simulation. In addition, multiple transverse modes or multiple longitudinal modes can be simulated.

Two choices of optical mode solver are available for edge-emitting lasers: `FEScalar` and `FEVectorial`. These solvers have been described in [Syntax of FE Scalar and FE Vectorial Optical Solvers on page 763](#).

Waveguide Optical Modes and Fabry–Perot Cavity

In an edge-emitting laser with a Fabry–Perot type cavity having high reflecting facets, the optical mode is invariant in the longitudinal direction. In this case, the optical propagation in the longitudinal z-direction can always be described by forward and backward traveling waves with characteristic $\exp(\pm i\beta z)$, where β is the longitudinal propagation constant.

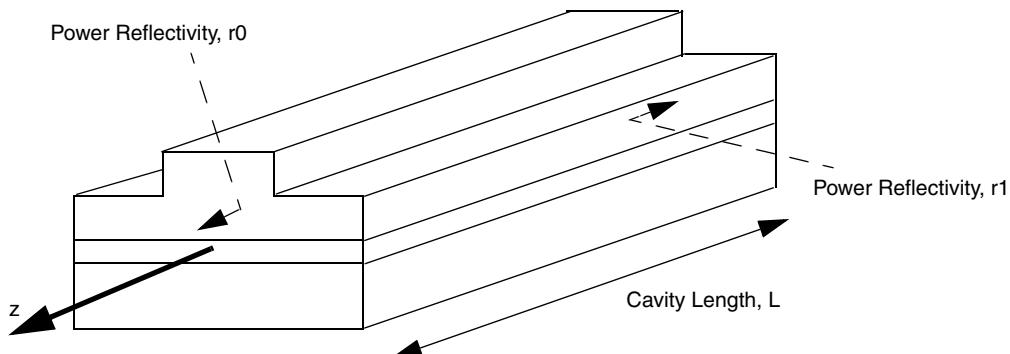


Figure 68 Waveguide problem in Fabry–Perot cavity assuming longitudinal invariance in z-direction

27: Lasers

Edge-emitting Lasers

Therefore, the vector wave equation reduces to the Helmholtz equation:

$$(\nabla_{xy} \times \nabla_{xy} \times -\nabla_{xy}(\varepsilon^{-1}\nabla_{xy} \cdot \varepsilon) + (\omega^2 \mu_0 \varepsilon(x, y) - \beta^2)) \cdot \mathbf{E}_t(x, y) = \mathbf{0} \quad (742)$$

where $\mathbf{E}_t(x, y)$ is the transverse vectorial electric field, ω is the angular frequency, ε is the permittivity, β is the complex propagation constant, and ∇_{xy} is the transverse gradient operator. The optical eigenmodes are the waveguide modes, which are characterized by the mode profile and propagation constant, β .

In a weakly guiding waveguide where the term caused by the refractive index inhomogeneity, $\nabla_{xy}(\varepsilon^{-1}\nabla_{xy} \cdot \varepsilon)\mathbf{E}_t(x, y)$, is small, the vector Helmholtz equation reduces to its scalar form:

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) \Psi + k_0^2(n^2(x, y) - \varepsilon_{eff})\Psi = 0 \quad (743)$$

where the scalar wavefunction $\Psi(x, y)$ describes either the TE-polarized or TM-polarized optical field component, ε_{eff} is the effective dielectric constant, $n(x, y)$ is the refractive index profile, and k_0 is the wavenumber of the mode. Sentaurus Device solves the vectorial and scalar Helmholtz equations by the finite-element method [4].

In the waveguide problem, the input parameters are:

- Wavelength (encompassed in $\lambda = (2\pi c / \omega) = 2\pi/k_0$)
- Refractive index profile $n(x, y)$
- Boundary conditions

The output is:

- Optical field profile
- Effective index (real part of propagation constant, $\text{Re}(\varepsilon_{eff}) = n_{eff} \cdot \frac{2\pi}{\lambda}$)
- Net optical loss (imaginary part of propagation constant)

The optical intensity is the square of the optical field, and it is normalized for use in the photon rate equation to compute the modal gain, spontaneous emission, and absorption loss (see Eq. 718–Eq. 720).

NOTE The imaginary part of the propagation constant is only included in the photon rate equation if the keyword `WaveguideLoss` is specified in the `Laser` statement.

Lasing Wavelength in Fabry–Perot Cavity

In a Fabry–Perot edge-emitting laser cavity, the wavelength is computed in the electronic solution and then passed to the Helmholtz equation solver. The physics is briefly outlined here. A Fabry–Perot cavity has eigenfrequencies:

$$\omega_p = \frac{\pi c}{\sqrt{\epsilon_r} \cdot L} p \quad (744)$$

where p is an integer containing the number of wavelengths that fit into the longitudinal cavity and L is the cavity length. In a Fabry–Perot cavity (typically a few millimeters), p is usually a very large number and the longitudinal mode spacing is very narrow (a few nanometers). The modal gain has a bandwidth of the order of 10 nm. Therefore, the lasing wavelength is computed as the cavity mode that has maximum modal gain $G(\omega)$ by varying p . Changing the wavelength changes the radiative recombination in the continuity equations. The default setting updates the lasing wavelength after each Newton iteration. This is a good approximation as long as the coupling between the electronic properties and the choice of wavelength is weak.

Alternatively, a self-consistent coupling is possible by including the keyword `Wavelength` in the `Plugin` statement:

```
Solve {...  
    quasistationary (...  
        Goal {name="p_Contact" voltage=1.6}  
    {  
        Plugin(BreakOnFailure){  
            Coupled {Electron Hole Poisson QWeScatter QWhScatter  
                PhotonRate}  
            Wavelength  
        }  
    }  
}
```

Output Power from a Fabry–Perot Laser

The output power at the facet 0 of the Fabry–Perot cavity is obtained by integrating the power flow (Poynting vector) in the z-direction over the facet surface [1]. This gives:

$$P_0 = S \frac{\hbar \omega c}{2 \sqrt{\epsilon_r} L} \ln\left(\frac{1}{r_0 r_1}\right) \frac{\sqrt{r_1}(1-r_0)}{(\sqrt{r_0} + \sqrt{r_1})(1 - \sqrt{r_0 r_1})} \quad (745)$$

27: Lasers

Edge-emitting Lasers

The power output from the two facets is given in the current file at the end of the simulation. The keywords for specifying the cavity length and facet power reflectivities are in [Table 182 on page 1150](#).

Symmetry Considerations

Imposing symmetry of the device in laser simulations requires the treatment of symmetry in both the optical and electrical parts of the problem. In particular, symmetry changes the boundary conditions for the optical solvers and must be handled carefully. See [Chapter 29 on page 761](#) for a detailed discussion about symmetry in the optics.

Conversely, symmetry in the electrical part of the problem is simpler. With the finite box method used in the simulation, the Neumann boundary condition is implicitly imposed at all boundaries that are not defined as the contacts. Therefore, there is no need for you to set the boundary specifically for symmetric structures.

However, an additional area factor of 2 must be specified in the Physics section:

```
Physics {...  
    AreaFactor = 2  
    Laser (...  
        Optics(  
            FEVectorial(...  
                Symmetry = Symmetric  
            )  
        )  
    )  
}
```

This adjusts the total electrode area and the optical output power, both of which double in the symmetric simulation mode.

Multiple Transverse Modes

A maximum of ten transverse modes are allowed. Different transverse modes have different spatial distribution of the optical density. Therefore, each mode experiences different modal gains, giving rise to different stimulated gain spectra. Assuming a Fabry–Perot cavity, the wavelength of each mode is given by the location of the peaks of the corresponding gain spectrum. In a DFB simulation, the wavelength is set and the modal gain of each mode is taken as the value of its respective gain spectrum at this wavelength.

Depending on the current flow distribution, the carriers available to contribute to the material gain of each mode are determined by carrier transport. The different distributions of spatial

optical intensity of different modes mean that the carriers are depleted (by stimulation recombination) at different locations, and this is commonly called the spatial hole-burning effect. You can view this effect by plotting the carrier density distribution in the active region of the device.

Traverse mode calculation is activated by specifying the keyword `TransverseMode`, which is the default and can be omitted.

Multiple Longitudinal Modes

The longitudinal mode simulation is only possible with Fabry–Perot-type cavities. If the keyword `LongitudinalModes` is set in the `Physics-Laser` section of the command file, a multilongitudinal mode calculation is performed. Sentaurus Device automatically selects an odd number of modes, which ensures one central (or main) mode and an equal number of modes, smaller and greater than the main mode frequency. The frequency of the main mode is determined as in the single-mode case by calculating the Fabry–Perot mode with maximum mode gain.

The spacing of the frequencies of the side modes is calculated according to the resonant frequencies in the Fabry–Perot cavity, which is in turn defined by the cavity length (keyword `CavityLength` in the `Physics-Laser` section). There is only one transverse mode associated with the multiple longitudinal modes.

For 2D simulations, this means that all the longitudinal modes share the same modal gain spectrum and, hence, the same carrier population for stimulated recombination. The multiple longitudinal-mode simulation is important for analyzing the suppression ratio of the side mode of the laser.

NOTE The longitudinal mode option cannot be used with the `DFB` option.

Sentaurus Device offers two choices of multimode simulation: transverse and longitudinal modes. Such a simulation can be performed by setting the keyword `Modenumber=<int>` to greater than one, and specifying one of the keywords `TransverseModes` (which is the default and can be omitted) or `LongitudinalModes` in the `Physics-Laser` section of the command file:

```
Physics {...  
  Laser (...  
    Optics (  
      FEVectorial (...)  
    )  
    # --- Choose transverse or longitudinal modes, but NOT both ---  
    TransverseModes  
    # LongitudinalModes
```

27: Lasers

Edge-emitting Lasers

```
ModeNumber = 5          # can choose up to 10 modes
...
)
}
```

For each longitudinal or transverse mode, one photon rate equation is solved.

NOTE You can select either multiple transverse modes or multiple longitudinal modes simulation, but not both simultaneously.

Specifying Fixed Optical Confinement Factor

The Helmholtz equation is not solved if the keyword optconfin=<float> is specified:

```
Physics {...  
  Laser (...  
    Optics (...  
      optconfin = 0.9  
    )  
  )  
}
```

In this case, the spatial optical intensity Ψ is assumed to be constant over the cavity, that is, the local mode gain and spontaneous emission at each active vertex are multiplied by the constant optconfin=<float> factor.

Simple Distributed Feedback Model

A simple distributed feedback (DFB) laser simulation can be performed by setting the keyword DFB in the Physics-Laser section of the command file:

```
Physics {...  
  Laser (...  
    Optics (  
      FEVectorial (...)  
    )  
    # --- Specify DFB simulation  
    TransverseModes  
    DFB (DFBPeriod=0.11)      # [micrometers]  
    CavityLength = 200         # [micrometers]  
    lFacetReflectivity = 0.3
```

```
rFacetReflectivity = 0.3
GroupRefract = 3.4      # fix effective index
...
)
}
```

The argument `DFBPeriod=<float>` specifies the period of the DFB grating, Δ , in units of μm . As a result, the lasing wavelength is assumed to occur at the Bragg wavelength, $\lambda_B = 2n_{\text{eff}}\Delta$, where n_{eff} is the effective refractive index. The effective index is solved from the Helmholtz equation using either the scalar (`FEScalar`) or vectorial (`FEVectorial`) optical solvers and can change if the refractive index profile changes in the simulation. The wave propagation in the longitudinal DFB grating, however, is not simulated. You can also set a fixed constant effective index with the keyword `GroupRefract=<float>`.

Bulk Active-Region Edge-emitting Lasers

Bulk active-region lasers can also be simulated. Similar to the case of the quantum well laser, you must specify the active material region by the keyword `Active`. In the `Physics-Laser` section of the command file, all the quantum well-related keywords should be removed: `QWTransport`, `QWExtension`, `QWScatModel`, `QWeScatTime`, `QWhScatTime`, `Strain`, and `SplitOff`. In this case, Sentaurus Device treats the active region as a bulk region, and the stimulated gain is computed as a bulk material gain. The carriers are assumed to scatter into the bulk active region by thermionic emission.

Leaky Waveguide Lasers

In many laser designs, the refractive index of the substrate or top layers is near the value of the guiding layer and is higher than that of the cladding layers. In such a case, any waves penetrating into the substrate or top layer will radiate outwards contributing to additional losses of the mode. Such leakage can also be used to discriminate higher order modes (which have higher losses) from the fundamental mode so that single-mode high-power laser diodes can be designed.

The `FEVectorial` optical solver coupled with PML (see [Perfectly Matched Layers on page 773](#)) makes Sentaurus Device an ideal tool to simulate such leaky waveguide lasers. In fact, Sentaurus Device has been successfully used to optimize leaky waveguide lasers for single-mode high-power operations [\[5\]](#).

Device Physics and Parameter Tuning

There are many quantities that you may want to optimize for the best laser diode performance or may want to tune in order to match experimental results. A selected list of these quantities and ways to tune them are:

Optical mode shape

The shape is controlled by the geometry and refractive index of the device. Sentaurus Workbench provides an automatic parameterization option for you to optimize geometric feature sizes, layer thickness, refractive index profile, and so on for the required mode shape.

Discretization of optical grid

The optical solvers use the finite-element method and a general rule is to use 20 points per wavelength to generate the mesh. This should provide an accurate solution for the optical mode.

Lasing threshold current

The threshold current is a function of the dark recombinations (Auger and SRH), optical losses, gain spectrum, and radiative recombination. The SRH lifetimes and Auger coefficients can be changed in the parameter file. Additional background optical loss can be input by `OpticalLoss=<float>` in the `Physics-Laser` section of the command file. The stimulated and spontaneous gain spectrum can be scaled by the keywords `StimScaling=<float>` and `SponScaling=<float>` in the `Physics-Laser` section of the command file.

Slope efficiency of L-I curve

This is primarily determined by the injection efficiency and the photon lifetime, which can be changed by the optical losses. However, changing the optical losses also affects the threshold current.

Temperature profile

The temperature distribution is sensitive to the boundary conditions specified at the thermodes. You can change the `SurfaceResistance` in the `Thermode` section of the command file to tune the temperature profile (see [Performing Temperature Simulation on page 859](#)).

Thermal rollover

The rollover is caused by self-heating effects of the laser diode. Possible major causes are Auger recombination and increased quantum well current leakage.

Vertical-Cavity Surface-emitting Lasers

Vertical-cavity surface-emitting lasers (VCSELs) are difficult devices to simulate due to their many layers (usually more than 100 layers). A full 3D simulation of a VCSEL is not feasible because it requires an excessive amount of computing resources.

To reduce the computational load, cylindrical symmetry is assumed so that the VCSEL problem is reduced to a quasi-2.5-dimensional problem. Each mesh on the (ρ, z) plane represents a solid ring rotated around the z -axis. As a result of such symmetry, the optical field can be expanded in cylindrical harmonics, and this further helps to reduce the size of the problem (see [Cavity Optical Modes in VCSELs](#)).

Three different types of optical solver are available to solve for the cavity modes of a VCSEL. There is one vectorial solver (`FEVectorial`) and two scalar solvers (`EffectiveIndex` and `TMM1D`). Only the vectorial solver provides accurate computation of the scattering and diffraction losses in a VCSEL of any type of geometry. This is important in computing the photon lifetime, which is a critical parameter in determining the slope efficiency of the light power output and the threshold current.

Nevertheless, the scalar solvers are extremely fast and most useful in the initial design phase of the VCSEL structure. In particular, the effective index method (keyword `EffectiveIndex`) can compute fairly accurate resonant wavelengths for strongly index-guided VCSELs, including oxide-confined VCSELs.

All VCSEL simulations in Sentaurus Device must be performed on two different grids: one for the electrical problem and one for the optical problem.

Cavity Optical Modes in VCSELs

The cavity eigenproblem deals with resonance in a cavity and is a difficult problem to handle for an arbitrary geometry and inhomogeneous cavity. Cavity resonance is defined as the condition whereby a wave can sustain itself in harmony inside the cavity. This means that even after undergoing multiple internal reflections inside the cavity, the optical waves at every point in the cavity are in phase with each other. If the cavity is leaky, some waves leak and the resonance decreases in amplitude and eventually diminishes. However, in a laser cavity, when stimulated emission of photons into the resonant mode is greater than the leakage, the resonance is sustained.

The cavity eigenproblem is then a statement of how to find these resonant modes (resonant wavelength) and the required gain within the active region that will balance the leakage (optical loss) to sustain resonance, that is, laser action.

27: Lasers

Vertical-Cavity Surface-emitting Lasers

Therefore, the cavity eigenproblem is different from the waveguide problem. In summary, the input is:

- Refractive index profile
- Boundary conditions

The required output for the cavity eigenproblem is:

- Resonant wavelength
- Resonant optical field profile
- Net optical loss

The detailed treatment of the cavity vectorial eigensolver for VCSELs in Sentaurus Device has been published [6] and only a summary of key ideas is presented here. The treatment essentially follows the original idea of Henry [1], which has been extended to a nonadiabatic form [7].

First, there is the time-dependent vector wave equation:

$$\nabla \times (\nabla \times \mathbf{E}(\mathbf{r}, t)) + \frac{1}{c^2} \frac{\partial^2}{\partial t^2} [\varepsilon_r(\mathbf{r}, t, \tau) \otimes \mathbf{E}(\mathbf{r}, t - \tau)] = -\mu_0 \frac{\partial^2}{\partial t^2} \mathbf{K}(\mathbf{r}, t) \quad (746)$$

where $\mathbf{K}(\mathbf{r}, t)$ is a source term caused by spontaneous emission that contributes to the electric field density. The term in the brackets is a time convolution of the dielectric function with the electric field.

The electric field is expanded (spectrally decomposed) into a discrete set of orthogonal modes:

$$\mathbf{E}(\mathbf{r}, t) = \sum_v a_v(t) e^{\int_0^t \omega'_v(\tau) d\tau} \Psi_v(\mathbf{r}, \omega) + cc \quad (747)$$

where $\Psi_v(\mathbf{r}, \omega)$ are vectorial modes, $a_v(t)$ are complex values, and cc are the complex conjugate terms. ω'_v is the frequency of the mode, a real valued function.

By substituting Eq. 747 into Eq. 746 and taking only the first-order terms of the time derivatives, a set of inhomogeneous equations is obtained:

$$\sum_v e^{\int_0^t \omega'_v(\tau) d\tau} \cdot \left[\left(\nabla \times (\nabla \times \Psi_v) - \frac{\omega'^2}{c^2} \varepsilon_r \Psi_v \right) \cdot a_v(t) + \tilde{\varepsilon}_r \Psi_v \cdot a_v(t) \right] = -\mu_0 \frac{\partial^2}{\partial t^2} \mathbf{K}(\mathbf{r}, t) \quad (748)$$

where:

$$\tilde{\varepsilon}_r = \frac{2i\omega'_v}{c^2} \left(\frac{\omega'_v}{2} \frac{\partial}{\partial \omega'_v} \varepsilon_r + \varepsilon_r \right) \quad (749)$$

In this case, the frequency dispersion of the dielectric function has been taken into account in [Eq. 749](#). To solve the above set of inhomogeneous equations, the auxiliary homogeneous equation is introduced:

$$\left[\nabla \times (\nabla \times \mathbf{E}) - \frac{\omega'_v}{c^2} \tilde{\epsilon}_r \right] \cdot \Psi_v = \omega''_v \tilde{\epsilon}_r \Psi_v \quad (750)$$

where ω''_v is defined to be the eigenvalue and the orthogonality relation is:

$$\int_{\text{volume}} \Psi_u \cdot \tilde{\epsilon}_r \Psi_v d\mathbf{r} = \delta_{uv} \quad (751)$$

At resonance, ω''_v must be purely real [6] and this forms the basis for finding the solution to the cavity eigenproblem. The frequency ω'_v is varied and [Eq. 750](#) is solved repeatedly for ω''_v . As ω'_v approaches the resonance value, the imaginary part of ω''_v approaches zero in an approximately linear manner. This gives you an advantage in accelerating the solution-hunting. Therefore, it is important that a ‘close enough’ target eigenvalue is provided by you to benefit from this advantage.

Next, to derive the photon rate equation for each cavity mode v , [Eq. 750](#) is substituted into [Eq. 748](#) and the orthogonality relation, [Eq. 751](#), is applied. After some manipulation, the cavity photon rate equation evaluates to the familiar form:

$$\frac{d}{dt} S_v = -2\omega''_v S_v + R_v^{\text{sp}} \quad (752)$$

where R_v^{sp} is the spontaneous emission rate. It transpires that the eigenvalue of [Eq. 750](#), $2\omega''_v$, is the net loss rate.

The material gain and absorption enter the photon rate equation indirectly through the dielectric function of the active region:

$$\epsilon_{r, \text{active}}(\mathbf{r}, \omega'_v) = \left(n(\mathbf{r}, \omega'_v) - \frac{c^2}{4\omega'_v} r^{\text{st}}(\mathbf{r}, h\omega'_v)^2 \right) + i \frac{n(\mathbf{r}, \omega'_v)c}{\omega'_v} r^{\text{st}}(\mathbf{r}, h\omega'_v) \quad (753)$$

$r^{\text{st}}(\mathbf{r}, h\omega'_v)$ is the stimulated emission coefficient and is discussed in [Radiative Recombination and Gain Coefficients on page 809](#). This approach ensures that the material gain and absorption are coupled rigorously between the electronics and optics. Therefore, the total cavity loss (or gain) is computed accurately. With the capability to model rapid changes in material gain, this cavity solver allows you to handle gain-guided VCSELs as well.

The photon rate equation is coupled to the Poisson, carrier continuity, and temperature (and hydrodynamic) equations using the Newton method, so the derivatives of $2\omega''_v$ with respect to a host of quantities for the Jacobian matrix entries are required.

27: Lasers

Vertical-Cavity Surface-emitting Lasers

$2\omega_v''$ can be identified as the relative change of the average electromagnetic energy stored in mode v :

$$2\omega_v'' = \frac{1}{\int_{\text{volume}} \langle W_v \rangle d\mathbf{r}} \cdot \int_{\text{volume}} \langle \frac{\partial W_v}{\partial t} \rangle d\mathbf{r} \quad (754)$$

where W_v is the energy density of the electromagnetic field of mode v derived from the Poynting vector. Assuming that the optical mode only changes slowly, the derivatives of $2\omega_v''$ can be computed from the derivatives of the dielectric function, $\epsilon_r(\mathbf{r}, t, \tau)$.

VCSEL Output Power

When the vectorial optical solver is used for VCSEL simulation, you must ensure that perfectly matched layers (PMLs) are used for the simulation structure. The PMLs act as wave absorbers to prevent reflections and artificially simulate the radiative boundary condition (see [Perfectly Matched Layers on page 773](#)). The output power emitted from the top surface is the time averaged dissipation of the optical power in the top PML.

The emitted power for mode v is, therefore, computed by the integral:

$$P_{\text{TopEmit}, v} = \int_{\text{vol - TopPML}} \langle \frac{\partial W_v}{\partial t} \rangle d\mathbf{r} \quad (755)$$

and takes into account the material absorption of the top PML. For purposes of calibration, it is possible to apply a scaling factor to the emitted power.

This factor can be specified in the `Physics-Laser-Optics` section of the command file as `PowerCouplingFactor=<float>`.

The optical model described above, however, does not take into account the reduction of the optical output power caused by perturbations of the optical cavity modes by free carrier absorption and other distributed internal optical losses. This is because this data is not fed into the optical solver that solves the optical eigenvalue problem, thereby determining the optical eigenmodes. To account for the reduced output power, a phenomenological correction can be used. In this case, the effective output power can be expressed as:

$$P_{\text{Total}, v} = P_{\text{TopEmit}, v} - x_{\text{FCA}} L_{\text{carr}} - x_{\text{bg}} L_{\text{bg}} \quad (756)$$

where L_{carr} is the free carrier absorption rate given by [Eq. 741, p. 702](#) and L_{bg} is the background loss of the cavity that can be specified with the command file keyword `OpticalLoss` in the `Physics-Laser` section.

The parameters x_{FCA} and x_{bg} can be specified in the Physics-Laser section:

```
Physics {...  
  Laser (...  
    ReducedOutcoupling(FreeCarrierAbsorption = <float> # xFCA  
                        OpticalLoss = <float>) # xbg  
  )  
}
```

Cylindrical Symmetry

Discretizing the VCSEL structure in 3D results in a prohibitively huge mesh size. Therefore, cylindrical geometry is assumed to reduce the size of the problem. By considering cylindrical symmetry in a body-of-revolution (BOR) about the symmetry axis, the cavity modes of the VCSEL can be further decomposed into cylindrical harmonics:

$$\Psi_v(\rho, \phi, z) = \sum_m \Psi_v^{(m)}(\rho, z) \cdot e^{im\phi} \quad (757)$$

It is well known that the cylindrical harmonics are orthogonal and, hence, the vectorial resonant optical field $\Psi_v^{(m)}(\rho, z)$ is solved for each cylindrical order, m , in 2D space. In the command file, the cylindrical harmonic order m is input using the keyword `AzimuthalExpansion`. (The syntax is discussed further in [FE Scalar Solver on page 764](#).)

Using experience from optical fiber modeling, the fundamental mode of an optical fiber is HE11, followed by its immediate higher order modes, TE01, TM01, and so on. This means that the fundamental mode has cylindrical harmonic order of $m = 1$, and the next two higher order modes have orders $m = 0$. You are encouraged to use different cylindrical harmonic orders to verify which one contains the fundamental resonant mode of the VCSEL cavity.

VCSEL structures are generally assumed to be cylindrically symmetric, and cylindrical symmetry in Sentaurus Device is managed in a slightly different way. The 2D plane whereby the device is drawn is treated as the (ρ, z) plane in cylindrical symmetry.

In addition to the specification of `Coordinates=Cylindrical` in the Physics-Laser-Optics-FEVectorial section, the keyword `Cylindrical` must also be added to the Math section:

```
Physics {...  
  AreaFactor = 1  
  Laser (...  
    Optics(  
      FEVectorial(...  
        Coordinates = Cylindrical  
      )
```

27: Lasers

Vertical-Cavity Surface-emitting Lasers

```
)  
VCSEL()      # specify this is a VCSEL simulation  
}  
}  
...  
Math {...  
    Cylindrical  
}
```

This is to ensure that the area and volume computations associated with each vertex is based on cylindrical symmetry.

NOTE In the cylindrical symmetry case, it is not necessary to specify an AreaFactor of 2.

Different Grid and Structure for Electrical and Optical Problems

A typical VCSEL structure contains over 100 layers. Sentaurus Device can manage such a large electrical problem, but the resultant Jacobian matrix may be too large for standard computers to handle. The carrier transport behavior in the distributed Bragg reflectors (DBRs) is not interesting with regard to the laser physics, and the DBRs will probably contribute only to an additional series resistance. Therefore, the size of the electrical problem can be simplified and reduced by replacing the DBR layers with an equivalent homogeneous bulk material with similar total resistance and thermal conduction properties. However, the actual layered structure must be used for the optical simulation because the optical resonance depends on the exact geometry of the VCSEL cavity.

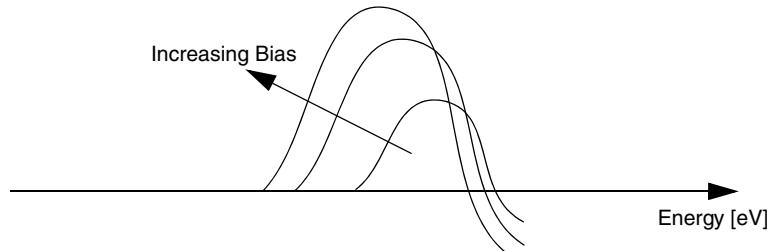
Therefore, there will be two different structures and grids for the electrical and optical problems. To handle this, the dual-grid mixed-mode capability of Sentaurus Device is used. The syntax for such a dual-grid simulation is described in [Simulating Vertical-Cavity Surface-emitting Laser on page 666](#).

For the vectorial optical (FEVectorial) solver, the optical mesh is discretized according to the requirements of the finite-element method. Adaptive meshing can be used to refine the mesh at sharp corners where scattering and diffraction effects are strong. For the scalar optical solvers (EffectiveIndex and TMM1D), the meshing of the optical grid can be relaxed: you will draw the VCSEL structure and apply coarse meshing to generate the optical grid.

Aligning Resonant Wavelength Within Gain Spectrum

Besides being a cavity problem, a VCSEL is different from an edge-emitting laser mainly in the size of the gain volume. The gain volume in a VCSEL is many times smaller than that of an edge-emitting laser. Therefore, ensuring a VCSEL lases is an intricate design task of minimizing the source of optical and electrical losses and maximizing the gain. One critical issue in VCSEL design is to align the resonant wavelength such that it is within the gain spectrum during lasing.

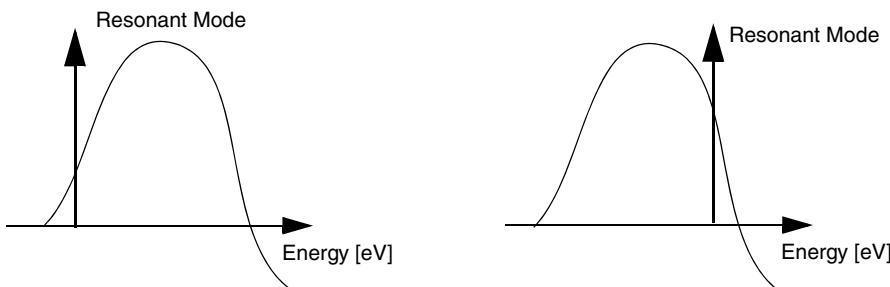
Self-heating of the VCSEL cavity causes red-shifts in both the gain spectrum (bandgap reduction as shown in [Figure 69](#)) and the resonant wavelength (thermal lensing effect). However, the shift in the gain spectrum is many times that of the resonant wavelength. Therefore, it is important to adjust the resonant wavelength such that it is near the peak of the gain spectrum at the required operating bias.



[Figure 69](#) As bias increases, the gain spectrum red-shifts (bandgap reduction) due to self-heating effects

[Figure 70](#) shows two starting positions of the resonant mode. As the bias increases, the gain spectrum shifts left, and the resonant mode of the model in [Figure 70 \(right\)](#) runs the risk of not obtaining enough gain required for lasing.

Therefore, it becomes a design issue of geometry and quantum well material gain to ensure that the resonant wavelength and gain spectrum (before lasing) resembles the model in [Figure 70 \(left\)](#).



[Figure 70](#) Resonant wavelength chosen to with different starting positions from the gain spectrum at very low bias (not lasing yet)

Approximate Methods for VCSEL Cavity Problem

Apart from the rigorous vectorial treatment of VCSEL cavity optics, Sentaurus Device also contains two approximate methods to compute the resonant modes in VCSEL cavities:

- Transfer matrix method for multilayers with a Gaussian transverse mode profile
- Effective index method

Both methods are scalar in nature and fast, and have low memory requirements. In particular, the transfer matrix method is suitable for users who want to look at how the transverse mode shape affects the different lasing characteristics of a VCSEL. The effective index method is best suited for index-guided VCSELs and it has been shown to compute accurate resonant wavelengths for many index-guided VCSEL structures including oxide-confined ones. When these approximate methods are used, Sentaurus Device uses the same photon rate equation as for edge-emitting lasers.

For further details about these solvers, see [Transfer Matrix Method for VCSELs on page 775](#) and [Effective Index Method for VCSELs on page 777](#).

Device Physics and Parameter Tuning

For a VCSEL, many quantities such as threshold current, slope efficiency, and stimulated and spontaneous gain can be similarly tuned as in the case of an edge-emitting laser. Other quantities that are of interest in a VCSEL simulation are:

Resonant wavelength selection

The resonant wavelength can be changed by changing the thickness or refractive indices of the DBR layers or the lambda cavity of the VCSEL. You can use the optics stand-alone option and the `EffectiveIndex` scalar solver to accomplish this optical design problem efficiently.

Aligning wavelength with gain spectrum

It is easier to tune the resonant wavelength rather than the gain spectrum. You are advised to run the simulation once to look at the gain spectrum shifts, then change the resonant wavelength as described in the previous paragraph.

Gain spectrum shifts

The band gap is reduced as temperature increases (due to self-heating). Higher carrier densities also result in bandgap renormalization caused by manybody effects. However, the bandgap renormalization shift is very small compared to the temperature bandgap shift.

Scattering and diffraction losses

The optical losses give the photon lifetime which is a critical parameter in the threshold current and slope efficiency. Only the vectorial optical (FEVectorial) solver can compute the optical losses accurately.

If you plan to use the scalar optical solvers (EffectiveIndex or TMM1D), it is advisable to run the vectorial optical solver once to obtain the accurate optical losses, then append an appropriate background loss (using keyword OpticalLoss in Physics-Laser section) or diffraction loss (using keyword DiffractionLoss in EffectiveIndex section) to the scalar optical solvers to enhance the accuracy of the scalar simulation.

References

- [1] C. H. Henry, “Theory of Spontaneous Emission Noise in Open Resonators and its Application to Lasers and Optical Amplifiers,” *Journal of Lightwave Technology*, vol. LT-4, no. 3, pp. 288–297, 1986.
- [2] R. K. Smith *et al.*, “Numerical Methods for Semiconductor Laser Simulations,” in *Fifth International Workshop on Computational Electronics (IWCE)*, Notre Dame, IN, USA, May 1997.
- [3] K. Rajkanan, R. Singh, and J. Shewchun, “Absorption Coefficient of Silicon for Solar Cell Calculations,” *Solid-State Electronics*, vol. 22, no. 9, pp. 793–795, 1979.
- [4] M. Koshiba, *Optical Waveguide Theory by the Finite Element Method*, Tokyo: KTK Scientific Publishers, 1992.
- [5] A. Witzig *et al.*, “Optimization of a Leaky-Waveguide Laser Using DESSIS,” in *3rd International Conference on Numerical Simulation of Semiconductor Optoelectronic Devices (NUSOD-03)*, Tokyo, Japan, October 2003.
- [6] M. Streiff *et al.*, “A Comprehensive VCSEL Device Simulator,” *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 9, no. 3, pp. 879–891, 2003.
- [7] G. A. Baraff and R. K. Smith, “Nonadiabatic semiconductor laser rate equations for the large-signal, rapid-modulation regime,” *Physical Review A*, vol. 61, no. 4, p. 043808, 2000.

27: Lasers

References

This chapter describes the physics and the models used in light-emitting diode simulations.

Modeling Light-emitting Diodes

From an electronic perspective, light-emitting diodes (LEDs) are similar to lasers operating below the lasing threshold. Consequently, the electronic model contains similar electrothermal parts and quantum-well physics as in the case of a laser simulation.

Therefore, the theory presented for quantum-well modeling in [Chapter 30 on page 803](#) is applicable to LED simulations as well. The key difference between an LED and a laser is a resonant cavity design for lasers that enhances the coherent stimulated emission at a single frequency (for each mode). An LED emits a continuous spectrum of wavelengths based on spontaneous emission of photons in the active region. However, an alternative design for LEDs with a resonant cavity – the resonant cavity LED (RCLED) – uses the resonance characteristics to cause an amplified spontaneous emission in a narrower spectrum to allow for superbright emissions.

The simulation of LEDs presents many challenges. The large dimension of typical LED structures, in the range of a few hundred micrometers, prohibits the use of standard time-domain electromagnetic methods such as finite difference and finite element. These methods require at least 10 points per wavelength and typical emissions are of the order of $1\text{ }\mu\text{m}$. A quick estimate gives a necessary mesh size in the order of 10 million mesh points for a 2D geometry. Alternatively, the use of the raytracing method approximates the optical intensity inside the device as well as the amount of light that can be extracted from the device. In many cases, a 2D simulation is not sufficient and a 3D simulation is required to give an accurate account of the physical effects associated with the geometric design of the LED.

Innovative designs such as inverted pyramid structures, chamfering of various corners, surface roughening, and drilling holes are performed in an attempt to extract the maximum amount of light from the device. The device editor Sentaurus Structure Editor is well equipped to create complex 3D devices and provides great versatility in exploring different realistic LED designs.

Photon recycling is important because most of the light rays are trapped within the device by total internal reflection. There are two types of photon recycling: for nonactive regions and for the active region. The nonactive-region photon recycling involves absorption of photons in the nonactive regions to produce optically generated electron–hole pairs, and these subsequently join the drift-diffusion processes of the general carrier population. The active-region photon

28: Light-emitting Diodes

Coupling Electronics and Optics in LED Simulations

recycling is more complicated, and there are two models: the simple photon-recycling model and the full photon-recycling model. In the simple photon-recycling model, you can set the amplified spontaneous emission, absorption, and absorption re-emission coefficients (see [Setting Up Simple Photon-Recycling Model on page 740](#)) independently. The net carrier recombination rate computed from the photon recycling feature can be iterated self-consistently with the electronic solver. As a result, the total spontaneous emission power of the active region will be modified and the effects of photon recycling affect the extraction efficiency.

The ultimate goal in LED design is white light emission. The most promising and cost-effective technology for producing white light is through the spectral conversion of blue emissions using a luminescent material. The typical luminescent material of choice is phosphor because it has a high conversion efficiency. An additional requirement is brightness, and this directly implies increased carrier injection. This also infers that the active region could be filled with an abundance of carriers for stimulated gain. The effect of the amplification of the spontaneous emission by the stimulated gain is incorporated into the full photon-recycling model in Sentaurus Device. Together with the spectral conversion capability, Sentaurus Device is well positioned to simulate white LEDs in a self-consistent and physics-based framework.

Coupling Electronics and Optics in LED Simulations

Similar to a laser simulation, an LED simulation solves the Poisson equation, carrier continuity equations, temperature equation, and Schrödinger equation self-consistently. [Figure 71](#) illustrates the coupling of the various equation systems in an LED simulation.

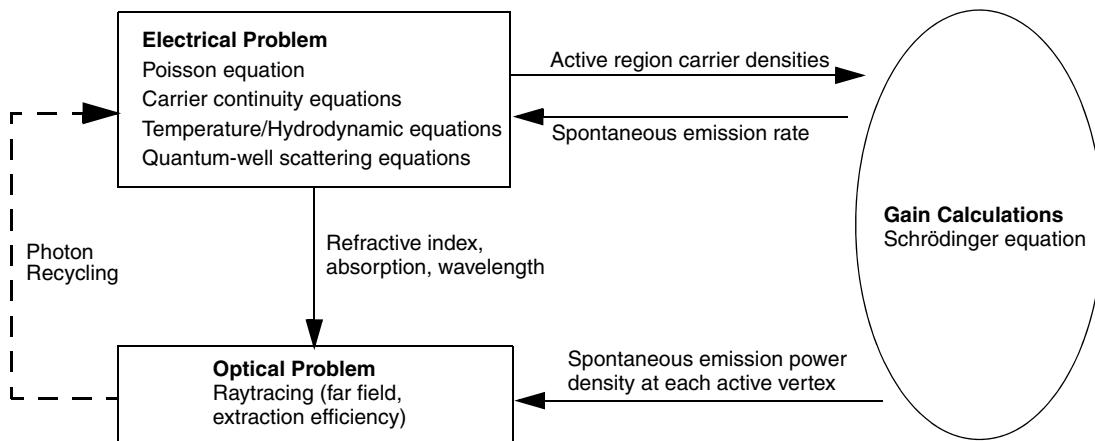


Figure 71 Flowchart of the coupling between the electronics and optics for an LED simulation

Discussion of LED Physics

Many physical effects are manifested in a LED structure. Current spreading is important to ensure that the current is channelled to supply the spontaneous emission sources at strategic locations that will provide the optimal extraction efficiency. When quantum wells are involved, Sentaurus Device computes a net carrier capture into the quantum wells based on scattering processes. In some cases, the LED structure gives a preferred polarization in the optical field and the spontaneous emissions in the active region can become anisotropic.

Changes to the geometric shape of the LED are made to extract more light from the structure. In most cases, the major part of the light produced is trapped within the structure through total internal reflection. As a result, the nonactive and active regions photon-recycling effect becomes relevant. This important physics has been incorporated into different physical models within Sentaurus Device. The nonactive photon-recycling (absorption of photons in nonactive regions) is switched on automatically by default.

Two active-region photon-recycling models can be chosen:

- The simple model where rays are multiplied with simple photon-recycling coefficients.,
- The full model where rays carrying spontaneous emission spectra undergo physics-based photon-recycling evolution.

Important aspects of LED design are easily simulated by Sentaurus Device. These include current spreading flow, geometric design, physics of quantum well transport, and extraction efficiency. There is also a feature that allows you to look at the wavelength spectrum of the far-zone radiation. This is especially useful in the design of white LEDs.

The LED simulator shares the same QW physics as the laser simulator, and you can use the full range of QW physics options (see [Chapter 30 on page 803](#)) in an LED simulation.

NOTE QW physics treatment is very complicated and depends strongly on the material system. The models used in one material system may not be applicable to others, and convergence cannot be guaranteed for all material systems.

LED Raytracing

Raytracing is used to compute the intensity of light inside an LED, as well as the rays that escape from the LED cavity to give the signature radiation pattern for the LED output. The basic theory of raytracing is presented in [Raytracing on page 407](#).

Arbitrary boundary conditions can be defined. A detailed description of how to set up the boundary conditions for raytracing is discussed in [Boundary Condition for Raytracing on page 422](#). This is particularly useful in 3D simulations where you can define reflecting planes to use symmetry for reducing the size of the simulation model.

In addition, you can use reflecting planes to take into account external components such as reflectors, an example of which is shown in [Figure 72](#).

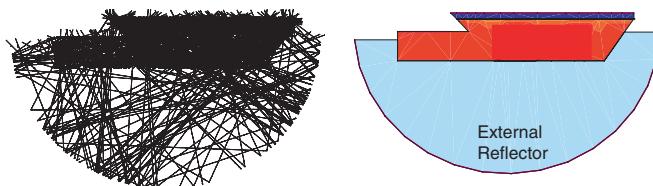


Figure 72 Using the reflecting boundary condition to define a reflector for LED raytracing

The raytracer needs to include the use of the `ComplexRefractiveIndex` model and to define the polarization vector. The syntax for this is:

```
Physics {...  
    ComplexRefractiveIndex (...  
        WavelengthDep ( real imag )  
    )  
    LED (...  
        Optics (  
            RayTrace(  
                PolarizationVector = Random # or (x y z) vector  
                ...  
            )  
        )  
    )  
}
```

When `PolarizationVector=Random` is chosen, random vectors that are perpendicular to the starting ray directions are generated and assigned to be the polarization vector of each starting ray. The direction of each starting ray is described in [Isotropic Starting Rays from Spontaneous Emission Sources](#) and [Anisotropic Starting Rays from Spontaneous Emission Sources on page 726](#).

Single-Grid Versus Dual-Grid LED Simulation

Both single-grid and dual-grid LED simulations are possible. However, in the case of a single-grid simulation, raytracing takes a longer time for the following reason: Raytracing builds a binary tree for each starting ray. Each branch of the tree corresponds to a ray at a mesh cell boundary. If the materials in two adjoining cells are different, the ray splits into refracted and reflected rays, creating two new branches. If the materials are the same in adjoining cells, the propagated ray creates a new branch. A fine mesh increases the depth of the branching significantly. Each new branch of the binary tree is created dynamically and if dynamic memory allocation of the machine is not sufficiently fast, the tree creation of the raytracing becomes a bottleneck in the simulation.

To overcome this problem, the grids for the electrical problem and raytracing problem are separated. The optical grid for raytracing is meshed coarsely. The binary tree created will then be smaller and raytracing is more efficient. Such a coarse mesh enables you to compute the extraction efficiency and output radiation pattern. However, the optical intensity within the device cannot be resolved well with a coarse mesh.

Isotropic Starting Rays from Spontaneous Emission Sources

The source of radiation from an LED is mainly from spontaneous emissions in the active region (this is further discussed in [Spontaneous Emission Power for LEDs on page 812](#)). The spontaneous emission in the active region of the LED is assumed to be an isotropic source of radiation and can be conveniently represented by uniform rays emitting from each active vertex, as shown in [Figure 73](#).

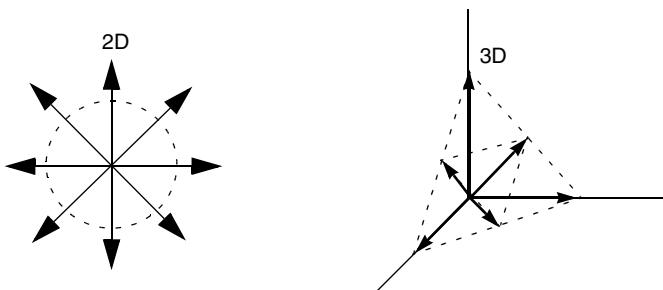


Figure 73 Uniform rays radiating isotropically from an active vertex source in 2D (left) and 3D (right) space: only one-eighth of spherical space is shown for the 3D case

Isotropy requires that the surface area associated with each ray must be the same. The isotropy of the rays in 2D space is apparent. In 3D space, achieving isotropy is not as simple as dividing

28: Light-emitting Diodes

LED Raytracing

the angles uniformly. The elemental surface area of a sphere is $r^2 \sin\theta(d\theta)(d\phi)$, so uniformly angular-distributed rays are weighted by $\sin\theta$ and, therefore, do not signify isotropy.

To approximate this problem in 3D, a geodesic dome approximation is used (the geodesic dome is not strictly isotropic). Rays are directed at the vertices of the geodesic dome. The algorithm starts by constructing an octahedron and, then, recursively splits each triangular face of the octahedron into four smaller triangles.

The first stage of this splitting process is shown in [Figure 73 \(right\)](#), where rays are directed at the vertices of each triangle. The minimum number of rays is six, that is, one is directed along each positive and negative direction of the axes. If the first stage of recursive splitting is applied, a few more rays are constructed as shown in [Figure 73](#), and the number of starting rays becomes 18. The second stage of recursive splitting gives 68 rays and so on. Therefore, you are constrained to selecting a fixed set of starting rays in the 3D case. Alternatively, you can input your own set of isotropic starting rays (see [Reading Starting Rays from File on page 728](#)).

Anisotropic Starting Rays from Spontaneous Emission Sources

In some LED designs, the geometry governs the polarization of the optical field in the device. The spontaneous gain is dependent on the direction of this polarization. Consequently, this leads to an anisotropic spontaneous-emission pattern at the source.

The anisotropic emission pattern is proposed to be described by the following parametric equations:

$$E_x = d1 \cdot \sin(\phi) + d4 \cdot \cos(\phi) \quad (758)$$

$$E_y = d2 \cdot \sin(\phi) + d5 \cdot \cos(\phi) \quad (759)$$

$$E_z = d3 \cdot \sin(\theta) + d6 \cdot \cos(\theta) \quad (760)$$

where the intensity is given by:

$$I = E_x^2 + E_y^2 + E_z^2 \quad (761)$$

The bases of sine and cosine are chosen based on the fact that the optical matrix element has such a functional form when polarization is considered (see [Polarization-dependent Optical Matrix Element on page 822](#)). By changing the values of d1 to d6, different emission shapes can be orientated in different directions, and this feature allows you to modify the anisotropy of the spontaneous emission.

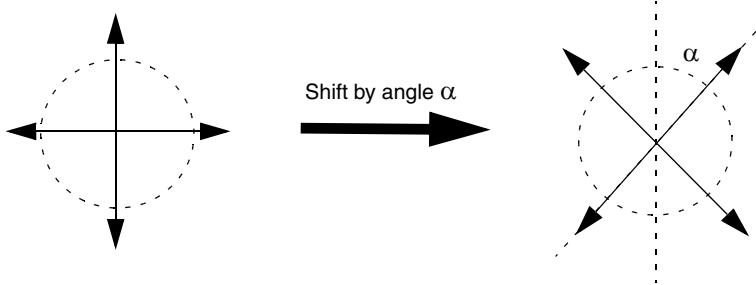
The syntax required to activate the anisotropic spontaneous emission feature is:

```
Physics {...  
    LED (...  
        Optics(...  
            RayTrace(...  
                EmissionType(  
                    #Isotropic          # default  
                    Anisotropic(  
                        Sine(d1 d2 d3)  
                        Cosine(d4 d5 d6)  
                    )  
                )  
            )  
        )  
    )  
}
```

Randomizing Starting Rays

Spontaneous emission is a random process. To take into account the random nature of this process and still ensure that the emission of the starting rays from each active vertex source is isotropic, a randomized shift of the entire isotropic ray emission is introduced.

This is best illustrated in [Figure 74](#) where only four starting rays are used for clarity. For each active vertex, a random angle is generated to determine the random shift of the distribution of the isotropic starting rays. The same concept is also used for the 3D case, and this gives a simple randomization strategy for using raytracing to model the spontaneous emissions.



[Figure 74](#) Shifting the distribution of entire isotropic starting rays by an angle α

NOTE Randomization of the starting rays is activated by the keyword `RaysRandomOffset` in the `RayTrace` statement. The default is a fixed angular shift, which is determined by the active vertex number.

Reading Starting Rays from File

The geodesic ray distribution is not truly isotropic. As a result, some percentages of error are incurred when isotropy is really needed. To circumvent this issue, a new feature to allow you to read in a set of source isotropic starting rays has been implemented. The syntax is:

```
Physics {...  
    LED (...  
        Optics (...  
            RayTrace (...  
                RaysPerVertex = 1000  
                SourceRaysFromFile("sourcerays.txt")  
            )  
        )  
    )  
}
```

It is important to note that `RaysPerVertex` specifies the number of direction vectors to read from the file. The input file contains 3D direction vectors for each starting ray; an example of which is `sourcerays.txt`:

```
-0.239117 0.788883 0.566115  
0.776959 0.548552 -0.308911  
-0.607096 -0.042603 0.793485  
-0.158313 -0.276546 -0.947871  
0.347036 -0.805488 0.480370  
...
```

There are several methods for generating 3D isotropic rays, for example, the constrained centroid Voronoi tessellation (CCVT). On the other hand, you can also use this option to import experimentally measured ray distribution profiles.

Moving Starting Rays on Boundaries

Starting rays from active vertices on boundaries create the problem that half of those rays are propagated directly out of the device. To alleviate this problem, a new feature is implemented to allow you to shift the starting position of such rays inwards of the device. Typical values used are from 1 to 5 nm. The syntax is:

```
Physics {...  
    LED (...  
        Optics (...  
            RayTrace (...
```

```

        MoveBoundaryStartRays(float)      # [nm]
    )
)
)
}
}
```

Debugging Raytracing

Rays are assumed, by default, to irradiate isotropically from each active vertex. In the case of quantum wells, an artifact of this assumption may cause unrealistic spikes in the radiation pattern. Consider those source rays that are directed within the plane of the quantum wells. These rays will mostly transmit out of the device at the ending vertical edges of the quantum wells. Realistically, the rays would have a much higher probability of being absorbed and re-emitted into another direction than traversing the entire plane of the quantum well.

To circumvent this problem, use the anisotropic emission as described in [Anisotropic Starting Rays from Spontaneous Emission Sources on page 726](#). Alternatively, one can try to exclude the source rays emitting within a certain angular range from the horizontal plane, that is, the plane of the quantum well. The power from these excluded rays will be distributed equally to the rest of the rays that are not within this angular range. This results in an approximate anisotropic emission shape at each active vertex. The syntax for this exclusion is:

```

Physics {
    ...
    LED (
        ...
        Optics (
            ...
            RayTrace (
                ...
                ExcludeHorizontalSource(<float>)    # in degrees
            )
        )
    )
}
```

To add more flexibility to LED raytracing, debugging features are implemented. These include:

- Setting a fixed observation center for the LED radiation calculations.
- Fixing a constant wavelength for raytracing.
- Allowing you to print and track the LED radiation rays (in a certain angular zone) back to its source active vertex.

These features are described in this syntax:

```

Physics {
    ...
    LED (
        ...
        Optics (
            ...
            RayTrace (
                ...
                ObservationCenter = (<float> <float>)
```

28: Light-emitting Diodes

LED Raytracing

```
                                # in micrometers, 3 entries for 3D
Wavelength = 888          # [nm] set fixed wavelength
DebugLEDRadiation(<filename> <StartAngle> <EndAngle>
                   <MinIntensity>)
    )
)
)
}
```

Print Options in Raytracing

The original `Print` feature of raytracing causes all ray paths to be printed. With multiple reflections or refractions and a large number of starting rays, the resultant image can become a black smudge. To reduce the number of ray paths that are printed, a `Skip` option is implemented within the `Print` feature. In addition, you can trace the ray paths originating from only a single active vertex. These features are described in this syntax:

```
Physics {
    LED (
        Optics (
            RayTrace (
                Print(Skip(<int>))           # skip printing every <int> ray paths
                Print(ActiveVertex(<int>))  # print only rays from active vertex
                                            # <int>
                PrintSourceVertices(<filename>)
                ProgressMarkers = <int>     # 1-100% intervals (integer)
            )
        )
    )
}
```

The option `PrintSourceVertices(<filename>)` outputs the list of active vertices, their global index numbering, and coordinates into the file specified by `<filename>`. If the number of source rays is large or the optical mesh is fine, raytracing takes some time to be completed. In this case, you can set the incremental completion meter by the keyword `ProgressMarkers = <int>`.

The `Print` option only outputs rays as lines with no other information. To obtain a raytree that contains intensity and other information, you can plot the raytree using the keyword `RayTrees` in the `Plot` section of the command file:

```
Plot {...  
    RayTrees  
}
```

The resultant raytree is plotted in TDR format and can be visualized by Tecplot SV, where each branch of the raytree can be accessed individually.

LED Radiation Pattern

Raytracing does not contain phase information, so it is not possible to compute the far-field pattern for an LED structure. Instead, the outgoing rays from the LED raytracing are used to produce the radiation pattern.

In 2D space, this is equivalent to moving a detector in a circle around the LED as shown in [Figure 75](#).

In 3D space, the detector is moved around on a sphere. The circle and sphere have centers that correspond to the center of the device. Sentaurus Device automatically determines the center of the device to be the midpoint of the device on each axis. Nonetheless, there is an option to allow you to set the center of observation (see [Debugging Raytracing on page 729](#) and [Table 196 on page 1158](#)).

To examine the optical intensity inside the LED, use `RayTraceIntensity` in the `Plot` statement.

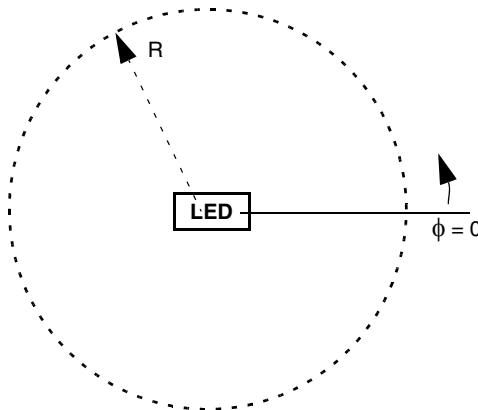


Figure 75 Measuring the radiation pattern in a circular path around LED at observation radius, R

The syntax required to activate and plot the LED radiation pattern is located in the `File`, `Physics-LED-Optics-RayTrace`, and `Solve-quasistationary` sections of the command file:

```
File {...
    # ----- Activate LED radiation pattern and save -----
    LEDRadiation = "rad"
```

28: Light-emitting Diodes

LED Radiation Pattern

```
}

...
Physics {...  
    LED (...  
        Optics (...  
            RayTrace(...  
                LEDRadiationPara(1000.0,180) # (radius_micrometers, Npoints)  
                ObservationCenter = (2.0 3.5 5.5) # set center  
            )  
        )  
    )  
}  
...
Solve {...  
  
    # ----- Specify quasistationary -----  
    quasistationary (...  
  
        PlotLEDRadiation { range=(0,1) intervals=3 }  
  
        Goal {name="p_Contact" voltage=1.8})  
        {...}  
    )
```

The LED radiation plot syntax works in the same way as the [GainPlot](#) (see [Plotting Gain and Power Spectra on page 855](#)) and the [OptFarField](#) plot (see [Far Field on page 786](#)) in the Quasistationary statement.

An explanation of this example is:

- The base file name, "rad", of the LED radiation pattern files is specified by `LEDRadiation` in the `File` section. The keyword `LEDRadiation` also activates the LED radiation plot.
- The parameters for the LED radiation plot are specified by the keyword `LEDRadiationPara` in the `Physics-Optics-RayTrace` section. You must specify the observation radius (in micrometers) and the discretization of the observation circle (2D) or sphere (3D).
- You can set the center of observation by including the keyword `ObservationCenter`. If this is not set, the center is automatically computed as the middle point of the span of the device on each axis.
- The LED radiation pattern can only be computed and plotted within the `Quasistationary` statement. The keyword `PlotLEDRadiation` controls the number of LED radiation plots to produce.

- The argument `range=(0, 1)` in the `PlotLEDRadiation` keyword is mapped to the initial and final bias conditions. In this example, the initial and final (goal) `p_Contact` voltages are 0 V and 1.8 V, respectively. The number of `intervals=3`, which gives a total of four ($= 3+1$) LED radiation plots at 0 V, 0.6 V, 1.2 V, and 1.8 V. In general, specifying `intervals=n` produces $(n+1)$ plots.
- If the LED structure is symmetric, the LED radiation is only computed on a semicircle.

The following sections briefly describe the files that are produced in the LED radiation plot for the 2D and 3D cases.

Two-dimensional LED Radiation Pattern and Output Files

Activating the LED radiation plot for a 2D LED simulation produces four different files (using the base name "rad"):

<code>rad.grd</code>	Discretized grid of a circle with polar coordinates (r, ϕ) in DF–ISE format.
<code>rad_00000_LEDRad.dat</code>	Data file containing the normalized radiation pattern in DF–ISE format to be used in conjunction with <code>rad.grd</code> .
<code>rad_00000_LEDRad.plt</code>	The normalized radiation pattern versus observation angle, which can be viewed in Inspect.
<code>rad_00000_LEDRad_Polar.grd</code>	The normalized radiation pattern projected onto a grid file and can be viewed in Tecplot SV. Run Tecplot SV and load the file <code>rad_00000_LEDRad_Polar.grd</code> . If the polar plot is not already shown, select Data > Alter > Transform Coordinates . Select the transformation Polar to Rectangular . Assign Source Theta = Y and Source R = x. Click Compute and click Close . The polar plot of the LED radiation pattern is then shown.

A sample output of the radiation plot of a 2D nonsymmetric LED structure is shown in [Figure 76 on page 734](#). The lower-left image corresponds to the file `rad_00000_LEDRad.plt` plotted by Inspect, and the right image is the product of the file `rad_00000_LEDRad_Polar.grd` plotted by Tecplot SV.

28: Light-emitting Diodes

LED Radiation Pattern

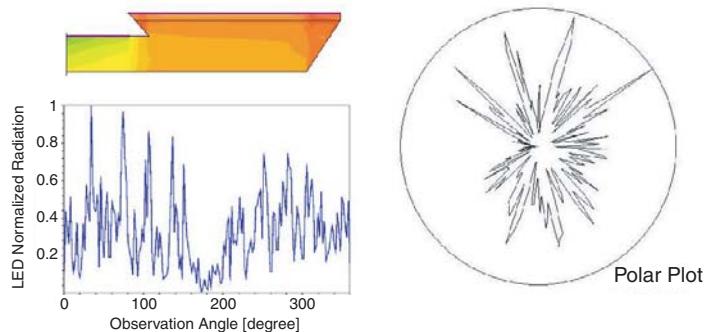


Figure 76 LED internal optical intensity (*upper left*), normalized radiation intensity versus observation angle (*lower left*), and polar radiation plot (*right*) computed by Sentaurus Device in 2D LED simulation

Three-dimensional LED Radiation Pattern and Output Files

There are two output files for the radiation pattern in the case of a 3D LED simulation:

`rad.grd`

Discretized grid of a sphere with spherical coordinates (r, ϕ, θ) in DF-ISE format.

`rad_000000_LEDRad.dat`

Data file containing the normalized radiation pattern in DF-ISE format to be used in conjunction with `rad.grd`. As Tecplot SV treats all grid input and displays the output as Cartesian coordinates, it is necessary to transform the data so that the spherical coordinates data can be viewed. This is performed with the same steps as in the 2D case: Run Tecplot SV and load the files `rad.grd` and `rad_000000_LEDRad.dat`. If the spherical contour map is not already shown, select **Data > Alter > Transform Coordinates**. Select the transformation **Spherical to Rectangular**. Assign Source Theta=Z, Source R=X, and Source Psi=Y. Click **Compute** and click **Close**. The spherical plot of the 3D LED radiation pattern is then shown. A sample of the radiation pattern of a 3D LED simulation is shown in [Figure 77 on page 735](#).

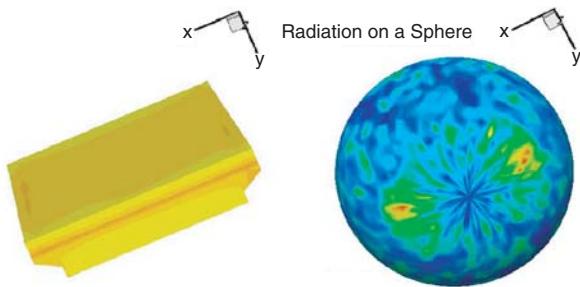


Figure 77 LED internal optical intensity (*left*) and normalized radiation intensity projected on a sphere (*right*) of 3D LED simulation

Spectrum-dependent LED Radiation Pattern

Unlike a laser beam with single-frequency emissions, rays emitting from an LED carry a spectrum of frequencies (or energies). Sentaurus Device monitors the spectrum of each ray as it undergoes the process of raytracing in and out of the device. The resultant spectrum of the LED radiation pattern can then be plotted.

To activate this feature, include the keyword `LEDSpectrum` in the command file:

```
Physics {...
    LED {...}
        Optics ...
            RayTrace{...
                LEDSpectrum(<startenergy> <endenergy> <numpoints>)
            }
        }
    }
}
```

This feature must be used in conjunction with the `LEDRadiation` feature so that the file names of the LED radiation plots and the observation angles can be specified. Other notable aspects of the syntax are:

- `<startenergy>` and `<endenergy>` give the energy range of the spectrum to be monitored. These parameters are floating-point entries with units of eV.
- `<numpoints>` is an integer determining the number of discretized points in the specified energy range.

Tracing Source of Output Rays

Optimizing the extraction efficiency is critical in an LED design. Other than modifying the shape of the device, you can use the fact that the rays originating from certain zones in the active region have a higher escape or extraction rate. By designing the shape of the contact to channel higher currents into these zones, the extraction efficiency can effectively be increased.

To facilitate the identification of such zones, a feature that correlates the output rays to their source active vertices is implemented. The intensity of each output ray is added to an `LED_TraceSource` variable at each active vertex. At the end of the simulation, a profile of `LED_TraceSource` is obtained, which provides an indication of the best localized regions to which more current can be channeled.

The activation of this feature requires keywords to be inserted into two different statements of the command file:

```
Plot { ...
    LED_TraceSource
}

Physics { ...
    LED ( ...
        Optics(
            RayTrace ( ...
                TraceSource()
            )
        )
    )
}
```

Nonactive Region Absorption (Photon Recycling)

The default setting for LED simulations includes the contribution of absorbed photons in nonactive regions to the continuity equation as a generation rate. This is the basic conception of the nonactive-region photon recycling. In most cases, LEDs are designed with larger bandgap material in nonactive regions to eradicate intraband absorption.

As a result, it may not be necessary to include very small values of nonactive absorption in some situations. To allow for such flexibility, nonactive absorption can be switched off as an option. The syntax is:

```
Physics { ...
    LED ( ...
        Optics ( ...
```

```

    RayTrace ( ...
        NonActiveAbsorptionOff
    )
)
)
}
}

```

If the keyword `NonActiveAbsorptionOff` is used, you would see zero entries in the `OpticalGeneration` plot variable.

Accelerating Gain Calculations and LED Simulations

The computation bottleneck in LED simulations with gain tables is the calculation of the spontaneous emission rate at every Newton step. The spontaneous emission rate (unit is s^{-1}) is an energy integral of the spontaneous emission coefficient (unit is $s^{-1}eV^{-1}cm^{-3}$), and the integration is performed as a Riemann sum. To accelerate this calculation, a Gaussian quadrature integration can be used instead. The syntax is:

```

Physics { ...
    LED ( ...
        Optics ( ...
            RayTrace ( ...
                GainTableSpeedup(GaussianQuadOrder = 5)
            )
        )
    )
}

```

The order can be from 1 to 20. This order refers to the order of the Legendre polynomial that is used to fit the spontaneous emission spectrum in the energy range of the integration. The Gaussian quadrature integration is exact for any spectrum that can be expressed as a polynomial.

In addition, you can speed up all gain calculations (with or without the gain tables) by specifying the following syntax in the `Math` section of the command file:

```

Math {...
    BroadeningIntegration ( GaussianQuadrature ( Order = 10 ) )
    SponEmissionIntegration ( GaussianQuadrature ( Order = 5 ) )
}

```

The Gaussian quadrature numeric integration is then used in all parts of the gain calculations including broadening effects.

NOTE Convergence issues can be experienced in some situations. In such cases, switch off the Gaussian quadrature integration for SponEmissionIntegration and BroadeningIntegration in the Math section.

Active-Region Photon Recycling

Two active-region photon-recycling models are available: the simple model and the full model. The simple model introduces you to the basic concepts of photon recycling by using fixed user-input parameters for amplified spontaneous emission (ASE) and absorption. When you are familiar with the concept of photon recycling, it is recommended to use the full photon-recycling model where the ASE and absorption parameters are automatically computed and are functions of the injection current.

NOTE The active-region photon recycling is useful only if you can model the gain spectra in the active region accurately.

Simple Photon-Recycling Model (Active Region)

To model physical effects of photon recycling with raytracing, three separate processes have been constructed in this simple model:

1. An ASE coefficient, ASE, representing the fraction of the ray that will cause stimulated emission in the active region. Therefore, a ray passing through the active region will be amplified by this ASE coefficient and the constraint is $\text{ASE} \geq 1$. If $\text{ASE} = 1.0$, there is no amplification. This process will deplete carriers in the active region.
2. An absorption coefficient, Abs, representing the fraction of the power of the ray that will be lost and this lost power will be converted into electron–hole pairs. These newly created carriers are then allowed to diffuse in the active region and spontaneously emit elsewhere or scatter out of the quantum well to join the leakage current.
3. An absorption re-emission coefficient, ReEmit, representing the fraction of the ray that will alter direction when the ray hits the active region. In this case, a new ray with a randomized direction is generated at the same spatial point. This inclusion is to allow for a possible change in the isotropy of spontaneous emission at the active vertices.

The constraint for the absorption and absorption re-emission coefficients is $\text{Abs} + \text{ReEmit} \leq 1$. The three processes are illustrated in [Figure 78 on page 739](#).

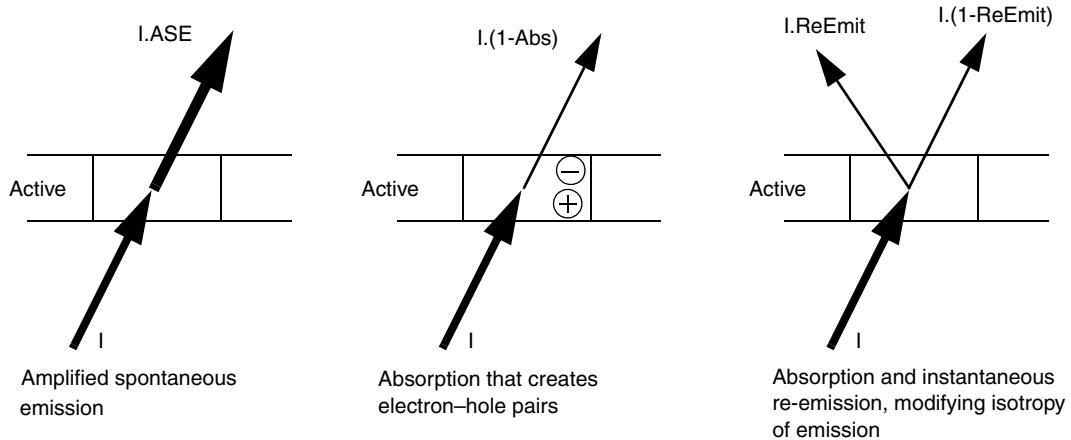


Figure 78 Modeling the photon-recycling effect by three separate processes

Combining the three processes yields the model that is implemented in Sentaurus Device (see Figure 79).

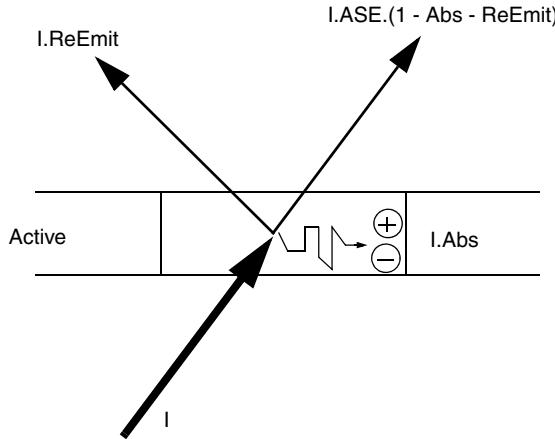


Figure 79 Simple photon-recycling model implemented in Sentaurus Device; there is no conservation of power since carriers can be generated or recombined by photon recycling

Each ray carries with it an optical spectrum corresponding to the active vertex from which it originates. With proper consideration, it is possible to ensure that the number of photons changed due to photon recycling corresponds exactly to the number of electron–hole pairs changed. ASE is a carrier recombination process, while photon absorption is a carrier generation process.

28: Light-emitting Diodes

Active-Region Photon Recycling

Adding the output rays and subtracting the input ray in [Figure 79 on page 739](#), the net recombination rate for this simple photon-recycling model is:

$$\begin{aligned} R_{\text{net}} &= N_{\text{ph}} \times [\text{ASE} \times (1 - \text{Abs} - \text{ReEmit}) + \text{ReEmit} - 1] \\ &= N_{\text{ph}} \times [(\text{ASE} - 1) \times (1 - \text{Abs} - \text{ReEmit}) - (\text{Abs})] \end{aligned} \quad (762)$$

where N_{ph} is the rate number of photons carried by the ray. N_{ph} starts off as the spontaneous emission rate in an active vertex. When the ray undergoes transmission, reflection, or absorption, N_{ph} reduces accordingly and becomes the rate number of photons associated with the ray. This net recombination rate is added to the continuity equations in the active region. Through this methodology, the simple photon-recycling model is iterated self-consistently with the electrical simulation (see [Figure 71 on page 722](#)). It is apparent from [Eq. 762](#) that the fraction of ASE power gained is $(\text{ASE} - 1) \times (1 - \text{Abs} - \text{ReEmit})$. This is added to the modified spontaneous emission power to give the total power of the device. The ASE power and spontaneous emission power, and their sum are output in the plot (.plt) file of the simulation.

Setting Up Simple Photon-Recycling Model

There are three steps in setting up the simple photon-recycling model:

- Define the photon-recycling contacts in the device editor, Sentaurus Structure Editor.
- Specify the ASE, absorption, and absorption re-emission coefficients.
- Use a `Plugin` statement to iterate the effects of active-region photon recycling with the electronic solver to achieve self-consistency.

Defining Photon-Recycling Contacts

In order to specify the various photon-recycling coefficients as described in the preceding section, the photon-recycling contacts are defined. These contacts can be drawn onto the device like the electrodes and thermodes, and then can be given a labeling name. Rays that traverse these photon-recycling (PR) contacts will be modified by the ASE, absorption, and absorption re-emission coefficients defined on these contacts.

After meshing, each contact is segmented into smaller edges for the 2D case or faces for the 3D case. Each of these PR edges or faces will be mapped to its neighboring active vertices so that any net recombination accumulated at this photon-recycling contact edge/face is distributed accordingly. In a dual-grid simulation setup, each PR edge/face in the optical grid is associated to the column of electrical active vertices that is bound by the transverse line/area of the PR edge/face. When a ray hits this PR edge/face, it will produce a net recombination (due to ASE and absorption, see [Figure 79](#) and [Eq. 762](#)). This net recombination is then distributed to the electrical active vertices that have been mapped to this PR edge/face.

For quantum wells, the photon-recycling contact is drawn as a boundary line in between the edges of the quantum wells of the optical device only. This boundary line can be formed by

breaking the QW region into two separate bodies. The line can be further partitioned into different sections by adding vertices along the line so that you can specify different photon-recycling coefficients on each section. An example is shown in [Figure 80](#). After meshing, each PR edge/face is mapped to its neighboring electrical active vertices. The mapping algorithm used is recursive and it searches for the active vertices, in the vicinity of the PR edge/face that are connected to the PR edge/face.

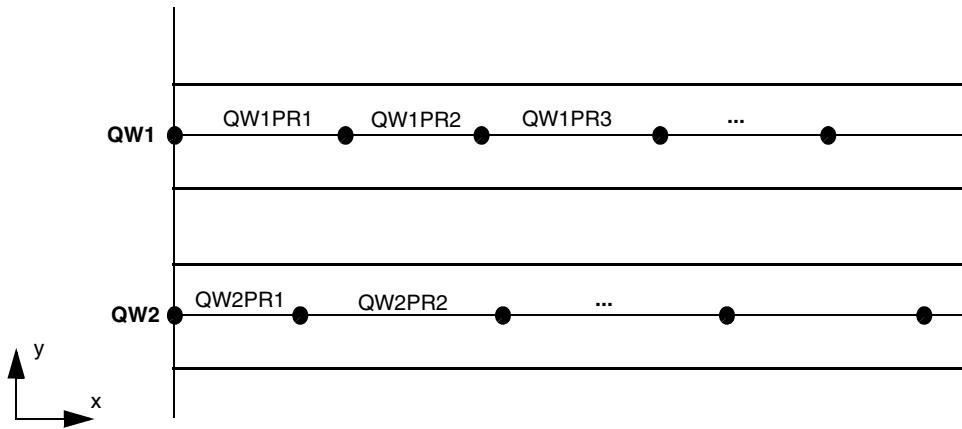


Figure 80 Drawing the photon-recycling contacts in quantum wells

For bulk active regions, use only a single PR contact to represent the entire active region. An example is shown in [Figure 81](#).

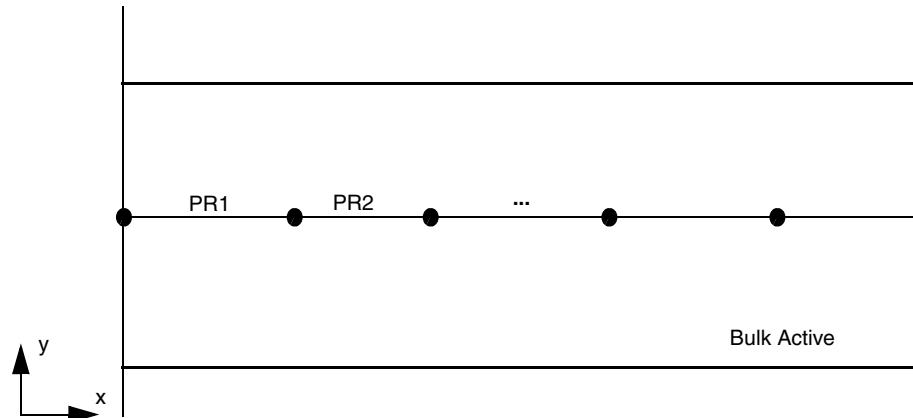


Figure 81 Using a single PR contact to represent an entire bulk active region; this PR contact is drawn in the middle of the bulk active region

28: Light-emitting Diodes

Active-Region Photon Recycling

In addition, you need to specify a new keyword in the command file to activate the special bulk active case:

```
Physics {...  
    LED (...  
        Optics( RayTrace (...  
            SimplePhotonRecycle(LumpActivePerEdge) # for 2D  
            SimplePhotonRecycle(LumpActivePerFace) # for 3D  
        )  
    )  
}
```

For example, in a 2D mesh, each PR edge is mapped to all the active vertices within the x-coordinate span of this edge. In the 3D mesh, each PR face is mapped to the active vertices that fall within the projected facial plane of this face.

By representing the active regions with PR contacts, the photon-recycling effect is approximated as an ensemble influence, and it is assumed that the ray of photons is modified by averaged photon-recycling coefficients. Since the various photon-recycling effects are based on ensemble-averaged probabilities of the carriers, this assumption makes a good approximation.

Specifying Photon-Recycling Coefficients

After the photon-recycling contacts have been drawn and labeled, they are specified in the command file with the following syntax:

```
RayTraceBC {  
    { Name="QW1PR1" PhotonRecycle = AbsEmit(0.2 0.2)  
        PhotonRecycle = ASE(1.02)  
    }  
    { Name="QW1PR2" PhotonRecycle = AbsEmit(0.1 0.2)  
        PhotonRecycle = ASE(1.1)  
    }  
    ...  
}
```

The two input parameters to `AbsEmit (<Abs> <ReEmit>)` are the absorption coefficient and the absorption re-emission coefficients, respectively. The obvious constraint to these two parameters is that their sum cannot be greater than 1.0.

Iterating Active-Region Photon-Recycling Effect Self-consistently

The effects of active-region photon recycling can be incorporated self-consistently into the electrical simulation by using the `Plugin` methodology:

```
Solve { ...
    Quasistationary { ...
        InitialStep = 0.001
        MaxStep = 0.1
        Minstep = 1e-6
        Goal { Parameter=drive.dc Value=1.5 }
    }
    {
        Plugin (BreakOnFailure) { ...
            Coupled { Electron Hole Poisson Contact Circuit
                QWeScatter QWhScatter }
            PhotonRecycle
        }
    }
}
```

Solving the coupled statement computes the spontaneous emission power in the active region. This power is then distributed accordingly to the raytree that was produced by raytracing. The rays that traverse the photon-recycling contacts will produce a net recombination rate at those contacts containing nonzero ASE or absorption coefficients. The net recombination rate is subsequently added to the continuity equations of the active region to ensure particle conservation. The coupled statement is solved again and the process is repeated until convergence is attained in all equations.

NOTE To switch on the active-region simple photon-recycling model, you must simultaneously have the `RayTraceBC{...}` statements and the `PhotonRecycle` keyword in the `Plugin` statement. To switch off the model, both must be removed.

Full Photon-Recycling Model (Active Region)

In the full model, physics-based photon-recycling effects are implemented by evolving the spontaneous emission spectrum carried by each ray [1][2]. Therefore, accurate modeling of gain and spontaneous emission spectra is required; otherwise, this feature will not be useful. Consider a ray that starts from an active mesh cell with a spontaneous emission spectrum. As it propagates within the LED, it will traverse another active cell that contains another stimulated spectrum. Both spectra will overlap each other, as illustrated in [Figure 82 on page 744](#).

28: Light-emitting Diodes

Active-Region Photon Recycling

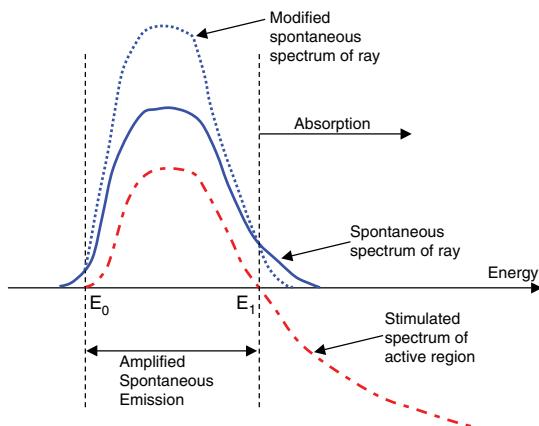


Figure 82 Spontaneous emission spectrum carried by a ray overlapping with stimulated emission spectrum of an active cell

In the active region of stimulated gain, the spontaneous emission rate at each energy level experiences stimulated emission and becomes amplified. Since amplification occurs over the continuous energy range of the gain, the result is amplified spontaneous emission (ASE): a carrier recombination process that contributes to the continuity equation. In the region of absorption, photons are converted into electron–hole pairs that are assumed to be immediately thermalized. A part of the absorbed photons (generated carriers) are assumed to be re-emitted instantaneously in a random direction, modifying the isotropic emission of that active cell. The other absorbed part is directly added to the continuity equation as a generation rate.

The net photon-recycling recombination rate is the difference between the ASE recombination and the carrier generation rates. Using this generalized physics-based photon-recycling model, the evolution of the spontaneous emission spectrum of each ray is tracked as it undergoes amplification and absorption processes through different traversals of the active regions. In other regions that are not designated as active, a simple absorption model is used but emissions in those regions are not allowed.

Suppose a ray originates from an active cell j with spontaneous emission rate (spectrum), $r_j^{\text{spon}}(E, \vec{r})$, and traverses cell i . The resultant spontaneous emission rate is modified according to the following rules:

$$r_i^{\text{spon}'}(E, \vec{r}) = \begin{cases} r_j^{\text{spon}} & , -\infty < E < E_0 \\ r_j^{\text{spon}} \cdot \exp(\Gamma_i G_i |\vec{n}|) & , E_0 < E < E_1 \\ r_j^{\text{spon}} \cdot \exp(-A_i |\vec{n}|) & , E_1 < E < \infty \end{cases} \quad (763)$$

where G_i and A_i are the energy-dependent gain and loss coefficients (units of cm^{-1}) of the optical intensity. G_i is the positive part of the stimulated gain spectrum whereas A_i is the negative part of the same stimulated spectrum. The absolute quantities of G_i and A_i are taken so that there is no ambiguity in the sign of the quantities in Eq. 763.

E_0 and E_1 are the zeros of the i^{th} stimulated spectrum, as shown in Figure 82 on page 744. Γ_i is the optical confinement factor that denotes the fraction of total power carried by this ray when it passes through the active cell i . In other words, the amount of stimulated emission is limited in each active cell, and the stimulated emission experienced by a ray must be weighted by the power (normalized to the total power) that it is carrying when it traverses the active cell. As the number of traversals increases, Γ_i is reduced and this prevents the infinite growth of power in the system.

The net photon-recycling recombination rate (unit is s^{-1}) at each traversal is given by:

$$R_{\text{net(PR)}} = \int \int_{-\infty}^{\infty} r_i^{\text{spon}'}(E, \hat{r}) dEd\hat{r} - \int \int_{-\infty}^{\infty} r_j^{\text{spon}}(E, \hat{r}) dEd\hat{r} \quad (764)$$

However, a fraction of the absorbed photons are permitted to be re-emitted instantaneously. This fraction is denoted by `ReEmit` and it must be accounted for in the net recombination rate:

$$R_{\text{net(PR)}} = \int \int_{E_0}^{E_1} (r_i^{\text{spon}'} - r_j^{\text{spon}}) dEd\hat{r} - (1 - \text{ReEmit}) \cdot \int \int_{E_1}^{\infty} (r_j^{\text{spon}} - r_i^{\text{spon}'}) dEd\hat{r} \quad (765)$$

It is clear that the contribution to the net recombination rate is from ASE recombination and the portion of photon absorption that was not re-emitted. The spectrum of the instantaneously re-emitted ray is replaced by the spontaneous spectrum of active cell i and is scaled appropriately. The direction of the re-emission ray is chosen randomly. Therefore, the conservation of particle creation and destruction in photons and carriers has been properly accounted for.

In the same spirit, you can derive the net power (unit is Js^{-1}) gained in each ray traversal of an active cell. Taking the energy and volume integral of Eq. 763 in different energy intervals, the following power components are obtained:

$$P_i = P_{\text{Spon}(j)} - P_{\text{Abs}(i)} + P_{\text{ReEmit}(i)} + P_{\text{ASE}(i)} \quad (766)$$

Summing the various power components of all the different rays that traverse the active cells, we obtain the ensemble total of absorption power, re-emission power, and ASE power. A detailed discussion of this theory can be found in the literature [2].

Setting Up Full Photon-Recycling Model

The steps for setting up the full photon-recycling model are similar to that of the simple model. Basically, the steps are:

- Define the photon-recycling contacts in the device editor, Sentaurus Structure Editor (see [Defining Photon-Recycling Contacts on page 740](#)).
- Specify the re-emission coefficients on these contacts.
- Include an additional statement in the Physics-LED-Optics-RayTrace section to specify the energy range of the spectrum carried by each ray.
- Use a Plugin statement to iterate the effects of active-region photon recycling with the electronic solver to achieve self-consistency.

The full photon-recycling model requires you to define photon-recycling (PR) contacts, as in the case of the simple photon-recycling model (see [Defining Photon-Recycling Contacts on page 740](#)). At these contacts, you must define the re-emission coefficient ReEmit, that is, the fraction of the absorbed photons that will be instantaneously re-emitted in a random direction.

Syntax for Full Photon-Recycling Model

The syntax for activating this model is:

```
RayTraceBC {  
    # Defining the re-emission coeff. of each PR contact  
    { Name="prcontact"  
        FullPhotonRecycle = ReEmit(0.2)  
    }  
}  
...  
Physics {...  
    LED (...  
        Optics (...  
            RayTrace (...  
                FullPhotonRecycle(  
                    # Defining energy range of spectrum carried by rays  
                    SpectrumEnergySpan = 1.0    # eV  
                    StartEnergy = 0.7          # eV user specify lower energy bound  
                    # Estimates of ASE and Absorption factors used  
                    # to build the initial ray tree  
                    EstimateASEFactor = 1.1   # greater than 1.0  
                    EstimateAbsFactor = 0.3   # between 0.0 and 1.0  
                    # Scales PR effect (power and recombination)  
                    PhotonRecycleScaling = 1.0  
                    SpectrumNumberOfPoints = 40  # default is 20  
                )  
            )  
        )  
    )  
}
```

```

        )
    )
}
...
Solve {...  

    Quasistationary (...)  

    { ...  

        Plugin(BreakOnFailure) {...  

            Coupled { Electron Hole Poisson Contact Circuit  

                      QWeScatter QWhScatter }  

            PhotonRecycle  

        }  

    }
}
}

```

In the Physics-LED-Optics-RayTrace section, you must specify the energy range of the spontaneous emission spectrum carried by each ray. The lower bound of the spectrum is automatically set at the bandgap energy minus an internally set shift to account for broadening effects.

To set the lower energy bound of the spectrum manually, you must include the keyword `StartEnergy`. In addition, you can specify the discretization of the spectrum that is carried by each ray by including the keyword `SpectrumNumberOfPoints`.

An initial raytree is first constructed and the photon-recycling effects are implemented by projecting the spectrum through the raytree. The photon-recycling effects (ASE and absorption) change with increases in bias current or voltage. For the raytree to provide adequate depth and span for the photon-recycling effects, you must specify the maximum expected absorption and ASE amplification of a ray that traverses an active region. These entries are denoted by `EstimateASEFactor` and `EstimateAbsFactor`.

You can scale the photon-recycling effects. The factor `PhotonRecycleScaling` is multiplied by the photon-recycling (ASE, absorption and re-emission) powers and recombination rates to alter the strength of photon recycling in the simulation. Such scaling can be also be used to adjust the transparency point. Other ways of altering the photon-recycling effects include changing the scaling factors for the spontaneous and stimulation emissions using `SponScaling` and `StimScaling`, respectively (see [Stimulated and Spontaneous Emission Coefficients on page 809](#)).

If the gain spectrum is plotted using the [GainPlot](#) option (see [Modal Gain as Function of Energy/Wavelength on page 856](#)), an additional modified spontaneous emission spectrum as a function of energy or wavelength is added to the existing spontaneous emission and stimulated emission spectra in the plot file. This modified spontaneous emission spectrum is the ensemble sum of all the evolved spectra carried by rays that escape from the LED structure, and it should be the type of spectrum typically measured experimentally.

The results of the full photon-recycling feature applied to a three-dimensional LED are shown in [Figure 83](#). As the bias current is increased, stimulated gain begins to dominate the spectral range of the spontaneous emission spectrum, leading to a significant increase in the ASE power.

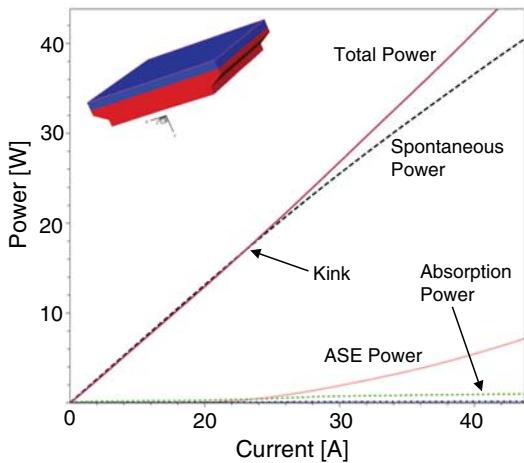


Figure 83 At higher current bias, ASE begins to contribute significantly to the total light power; (*inset*) 1x1 mm 3D LED used for simulation, with electrodes covering the entire top and bottom surfaces

Plotting Evolving Spectra

The evolving spectra of the full photon-recycling option can be plotted. This feature can only be activated in conjunction with the GainPlot feature. The evolving spectra are only printed at the designated time points as requested in the GainPlot statement in the Solve section. The syntax is:

```
Physics {...  
    LED (...  
        Optics (...  
            RayTrace (...  
                PlotEvolvingSpectra (  
                    Range = (1.2 1.8)          # eV  
                    NumberOfPoints = 40  
                    # Number of evolution levels to plot  
                    PlotLevel = 10  
                    # Activate plot of aggregate total evolving spectra  
                    PlotAverage()  
                    # Activate plot of evolving spectra of specific ray trees  
                    # Each ray tree is referred to by its starting ray index  
                    # Max of 100 ray trees allowed
```

```

        # The spectral density is plotted
        PlotRays(500 1000 1500 2000 2500 3000)
    )
)
)
)
}
}
```

If the gain-plot base file name is `gain`, the evolving spectra output files will be `d1_gain_gain_des_000000_evolved.plt`, `d1_gain_gain_des_000001_evolved.plt`, and so on. The spectra are scaled by the relative intensity of the ray at that photon-recycling level, and the general unit for these spectra should be $\text{s}^{-1}\text{eV}^{-1}\text{cm}^{-3}$. The photon-recycling level indicates the number of passes of the ray in the active region such that it experiences a photon-recycling event.

You can also print the list of starting rays so that you will know the index of the raytree to use above in the keyword `PlotRays(...)`. The output information includes the starting ray index, the starting position, the direction, and the spontaneous emission value. Similarly, this information is saved to the file whenever a `GainPlot` is called because there could be changes in the randomized ray directions if a completely new raytree is created and the spontaneous emission power changes. The syntax is:

```

Physics {...}
LED {...}
Optics {...}
RayTrace {...}
PrintRayInfo("rayinfo.txt")
}
)
}
}
```

If the ray information base file name is `rayinfo.txt`, the output files will be `rayinfo_000000_des.txt`, `rayinfo_000001_des.txt`, and so on.

Spectral Conversion

The most cost-effective technology for producing white light is through spectral conversion [2] of blue LED emissions using a luminescent material that encases the entire LED structure. Assume that the luminescent material has an absorption profile, $A_{\text{LM}}(E, \vec{r})$ defined in $[E_{a0}, E_{a1}]$ (unit is cm^{-1}) and an emission profile, $r_{\text{em}}(E, \vec{r})$, defined in $[E_{\text{em}0}, E_{\text{em}1}]$ (unit is $\text{s}^{-1}\text{eV}^{-1}\text{cm}^{-3}$).

28: Light-emitting Diodes

Spectral Conversion

A ray carrying a spectrum, $r_j^{\text{spon}}(E)$, that traverses the luminescent region will be modified by the absorption process, giving:

$$r_j^{\text{spon}'}(E, \hat{r}) = \begin{cases} r_j^{\text{spon}}(E, \hat{r}) \cdot e^{-A_{\text{LM}}|\hat{r}|} & , \quad E_{a0} < E < E_{a1} \\ r_j^{\text{spon}}(E, \hat{r}) & , \quad \text{otherwise} \end{cases} \quad (767)$$

Consequently, the absorption rate (unit s^{-1}) is:

$$R_{\text{absorbed(SC)}} = \int_{\hat{r}=-\infty}^{\infty} \int \{r_j^{\text{spon}}(E, \hat{r}) - r_j^{\text{spon}'}(E, \hat{r})\} dEd\hat{r} \quad (768)$$

Introducing the spectral conversion (quantum) efficiency, the photon emission rate (unit s^{-1}) due to spectral conversion is:

$$R_{\text{em(SC)}} = \text{ConversionEfficiency} \times R_{\text{absorbed(SC)}} \quad (769)$$

This value is used to scale the emission profile correctly. Adding the scaled emission profile to Eq. 767 gives the spectrally converted spectrum (unit $\text{s}^{-1} \text{eV}^{-1} \text{cm}^{-3}$):

$$r_{SC}^{\text{spon}}(E, \hat{r}) = r_j^{\text{spon}'}(E, \hat{r}) + \alpha_{\text{SC}} \cdot r_{\text{em}}(E, \hat{r}) \quad (770)$$

where the scaling factor, α_{SC} , is defined as:

$$\alpha_{\text{SC}} = \frac{R_{\text{em(SC)}}}{\int_{\hat{r}=E_{\text{emf}}}^{\infty} \int r_{\text{em}}(E, \hat{r}) dEd\hat{r}} \quad (771)$$

After spectral conversion, the ray is randomly emitted in another direction. These spectrally converted rays may re-enter the device and undergo additional photon-recycling cycles.

Defining Absorption and Emission Profiles

To simplify the possible multiple transitions of energy levels, the absorption profile can be input as a series sum of Gaussian functions and Lorentzian functions, or as a numeric table; likewise for the emission profile.

The Gaussian function is defined as:

$$f(x) = \text{DCoffset} + \frac{I_{\text{peak}}}{\sigma \sqrt{2\pi}} \cdot \exp\left(\frac{-(x-x_0)^2}{2\sigma^2}\right) \quad (772)$$

and the Lorentzian function is:

$$f(x) = DCoffset + \frac{1}{\pi} \cdot \frac{I_{peak} \cdot \left(\frac{W}{2}\right)}{(x - x_0)^2 + \left(\frac{W}{2}\right)^2} \quad (773)$$

Note that a `DCoffset` is added to shift the Gaussian or Lorentzian function. In addition, three parameters are introduced for extra flexibility:

- `ConversionEfficiency` (fraction of absorbed photons that will be converted to emitted photons)
- `AbsorptionScaling`
- `EmissionScaling`

The syntax in the parameter file is:

```
Material = "LuminescentMat" {
    ...
    SpectralConversion {
        ConversionEfficiency = float
        AbsorptionScaling = float
        EmissionScaling = float
        * lambda, sigma, width are in units of nm
        * peakvalue, dc_offset, absorption and emission are in units of cm^-1
        Analytic (
            AbsorptionProfile =
                Gaussian(lambda0 sigma peakvalue dc_offset lambda_range0
                         lambda_range1)
                Lorentzian(lambda0 width peakvalue dc_offset lambda_range0
                          lambda_range1)
            ...
        )
        EmissionProfile =
            Gaussian(lambda0 sigma peakvalue dc_offset lambda_range0
                     lambda_range1)
            Lorentzian(lambda0 width peakvalue dc_offset lambda_range0
                      lambda_range1)
        ...
    )
    NumericalTable (
        AbsorptionProfile =
            lambda0 value0
            lambda1 value1
        ...
    )
    EmissionProfile =
        ...
}
```

28: Light-emitting Diodes

Spectral Conversion

```
lambda0 value0
lambda1 value1
...
)
)
}
}
```

When multiple Gaussian and Lorentzian functions are specified in `AbsorptionProfile` and `EmissionProfile`, the final profile is the sum of these functions.

Spectral Conversion Syntax

The syntax for activating the spectral conversion feature is:

```
OpticalDevice OptSolver {...  
    # Spectral conversion material defined in optical grid only  
    Physics (region="luminescentmat") {  
        SpectralConversion(Analytic)  
        #SpectralConversion(NumTable)      # cannot be both analytic and  
                                            # numtable simultaneously  
    }  
}  
Dessis diode {...  
    Physics {...  
        LED (...  
        Optics (...  
            RayTrace (...  
                FullPhotonRecycle (...)  
                ActivateSpectralConversion (ScatterFactor = float)  
                                            # probability of scattering  
            )  
        )  
    }  
}
```

You must indicate whether to use the analytic functions or the numeric table in the `Physics` section of the luminescent region.

The full photon-recycling feature, `FullPhotonRecycle(...)`, must be simultaneously activated for the spectral conversion feature to work.

The probability of whether the spectrally converted ray will be given a new random direction is controlled by the parameter, `ScatterFactor`. A random number is first generated and if this random number is greater than `ScatterFactor`, no scattering occurs. Otherwise, another

random number is generated between $[-\pi, \pi]$ to change the direction of the transmitted ray. If `ScatterFactor=0`, the ray does not change direction. If `ScatterFactor=1.0`, the randomization of direction is definite. You can choose this random scattering option or Henyey–Greenstein scattering.

Henyey–Greenstein Volume Scattering Probability

Phosphor is commonly used as the luminescent material to spectrally convert blue/green emissions to a white tone. Different types of phosphor contain grain size particles that will scatter photons in some preferred direction. The angular scattering probability can be modeled using the Henyey–Greenstein scattering function.

The Henyey Greenstein scattering probability for each angle, θ , is:

$$p_{HG}(\theta) = \frac{1}{4\pi} \cdot \frac{1-g^2}{(1+g^2 - 2g\cos\theta)^{\frac{3}{2}}} \quad (774)$$

where g is the asymmetry factor of the big particle distribution. You can input g that spans [0,1]. The probability density function (PDF) is:

$$f(\theta) = p_{HG}(\theta) \cdot 2\pi \sin\theta \quad (775)$$

To map $f(\theta)$ to a uniform random number (x) distribution, $h(x)$, spanning [0,1], it is required that:

$$f(\theta)d\theta = h(x)dx \quad (776)$$

Working through the math gives the inverse function:

$$\theta = \arccos\left\{\frac{1}{2g} \cdot \left[1 + g^2 - \frac{(1-g^2)^2}{(1+g-2gx)^2}\right]\right\} \quad (777)$$

A uniformly distributed random number x spanning [0,1] is generated, and it is substituted into Eq. 777 to compute the eventual scattering angle, θ . When θ is computed, the transmitted ray direction is rotated using the Rodrigues rotation matrix:

$$\mathfrak{R} = \begin{bmatrix} \cos\theta + w_x^2(1-\cos\theta) & w_xw_y(1-\cos\theta) - w_z\sin\theta & w_y\sin\theta + w_xw_z(1-\cos\theta) \\ w_z\sin\theta + w_xw_y(1-\cos\theta) & \cos\theta + w_y^2(1-\cos\theta) & -w_x\sin\theta + w_yw_z(1-\cos\theta) \\ -w_y\sin\theta + w_xw_z(1-\cos\theta) & w_x\sin\theta + w_yw_z(1-\cos\theta) & \cos\theta + w_z^2(1-\cos\theta) \end{bmatrix} \quad (778)$$

where $\hat{w} = (w_x, w_y, w_z)$ is a unit vector of the rotation axis.

28: Light-emitting Diodes

Interface to LightTools®

The polarization vector of the transmitted ray is also appropriately resolved after the new ray direction has been computed. The syntax to activate Henyey–Greenstein scattering is:

```
Physics { ...
    LED ( ...
        Optics ( ...
            RayTrace ( ...
                MonteCarlo
                ActivateSpectralConversion(
                    HenyeyGreensteinScattering(float)    # g (0~1)
                )
            )
        )
    )
}
```

The parameter to the `HenyeyGreensteinScattering` is the asymmetry factor, g , and it has a span of [0,1]. In the spectral conversion feature, rays traversing the luminescent material inevitably alter the effective wavelength. In such cases, you can input an effective wavelength for each luminescent region. After a ray passes through each luminescent region, it uses the refractive index profile of the effective wavelength of that luminescent region. The syntax to specify effective wavelength for each luminescent region is required in the parameter file:

```
SpectralConversion {
    EffectiveWavelength = float    * [nm]
}
```

Since this model is a scattering event with a probability that depends on the initial impact angle, it must be used in conjunction with Monte Carlo raytracing.

Interface to *LightTools*®

LightTools® from Optical Research Associates (ORA®) is a Monte Carlo–type raytracer. It accepts a list of source rays as input, and it can optimize the packaging design for LEDs.

An interface has been built in Sentaurus Device to output rays from an LED device simulation that can be input into *LightTools* as source rays. This feature requires the LED radiation plot to be activated simultaneously so that the rays can be output at various quasistationary and transient states. The syntax for activating this feature is:

```
Physics { ...
    LED ( ...
        Optics ( ...
            RayTrace ( ...
                LEDRadiationPara(...
                PrintORArays(
```

```
    FileName = "string"
    SplitSpectrumRays = integer
    RefractiveIndexDelta = float
    NumberOfSpectralPoints = integer
    RangeOfSpectrum = (float float)
    VectorUnits = Micron      # or Millimeter
)
)
)
)
}
```

Each output ray is allowed to be split into several rays indicated by the parameter `SplitSpectrumRays`. These split rays will be given the weights of the spectrum sampled at different wavelengths. If a wavelength-dependent refractive index has been specified, the direction of each split ray will be perturbed by a small change in angle, given by the following formula:

$$d\theta = -\left(\frac{dn}{n}\right) \cdot \tan\theta \quad (779)$$

This formula is derived by taking small changes in Snell's law: $n \cdot \sin\theta = \text{constant}$. If the refractive index is not wavelength dependent, the wavelength changes will be a fraction of the parameter `RefractiveIndexDelta`.

NOTE In Sentaurus Device, raytracing and wavelength dependency of the refractive index can only be specified by using the complex refractive index model.

The other parameters of `NumberOfSpectralPoints` and `RangeOfSpectrum` define the output spectrum to be printed.

Two specific files are produced: one with the extension `_rays.txt` containing the ray information, and one with the extension `_spectrum.txt` containing the spectrum information.

The ray information consists of the position vector (x,y,z), the direction cosine (x,y,z), and the relative intensity. The spectrum information is indicated by the wavelength and relative intensity. Both files contain identifying headers that are required by *LightTools*. Sample formats of the two files are presented.

Example: orainterface_rays.txt

```
# Synopsys Sentaurus Device to ORA® LightTools®
# Ray Data Export File
DATANAME: SentaurusDeviceRayData
LT_FAR_FIELD_DATA: NO
LT_DATA_ORIGIN: 0 0 0
LT_RADIANC_FLUX: 1.000
LT_STARTOFDATA
-35.3165701 -0.8510182 11.5477636 0.7179369 0.5564080 -0.4183022
6.2207040e-08
-35.3165701 -0.8510182 11.5477636 0.9423966 -0.0113507 -0.3343049
4.6799284e-08
...
LT_ENDOFDATA
```

Example: orainterface_spectrum.txt

```
# Synopsys Sentaurus Device to ORA® LightTools®
# Aggregate Ray Spectrum Data Export File
DFAT 1.0
DATANAME: SentaurusDeviceSpectrum
CONTINUOUS
RADIOMETRIC
8.3369229e+02 0.0952224
8.4831847e+02 0.1650206
8.6346702e+02 0.2883805
8.7916642e+02 0.4638275
8.9544728e+02 0.6425363
9.1234251e+02 0.8041440
9.2988755e+02 1.0000000
9.4812064e+02 0.8919842
9.6708306e+02 0.5533345
9.8681945e+02 0.0430633
...
```

Device Physics and Tuning Parameters

Unlike a laser diode, an LED does not have a threshold current. Therefore, the carriers in the active region are not limited to any threshold value. This means that the spontaneous gain spectrum continues to grow as the bias current increases. The limiting factor for the growth is when the quantum well active region is completely filled and leakage current increases significantly.

There are two main design concerns for an LED:

- Extraction efficiency. This is mainly a problem of the geometric shape of the LED structure. Many LED structures have tapered sidewalls to help couple more light out of the device. The slope of the taper can be set as a parameter using Sentaurus Workbench, and the automatic parameter variation feature can be used to optimize the extraction efficiency of the LED geometry.
- Uniform current spreading. It is desirable to spread the current uniformly across the entire active region so that total spontaneous emissions can be increased. In Sentaurus Device, there is an option to switch off raytracing in an LED simulation. Switching off raytracing only forgoes the extraction efficiency and radiation pattern computation; the total spontaneous emission power is still calculated. This can assist you in the faster optimization of the LED device for uniform current spreading.

Modeling Organic Light-emitting Diodes

A brief introduction to organic devices and the available organic models are presented in [Chapter 13 on page 389](#).

To clearly demonstrate how the combination of different models in Sentaurus Device can be composed to model organic devices, an organic light-emitting diode (OLED) is used as an example. [Figure 84](#) shows a typical OLED structure.

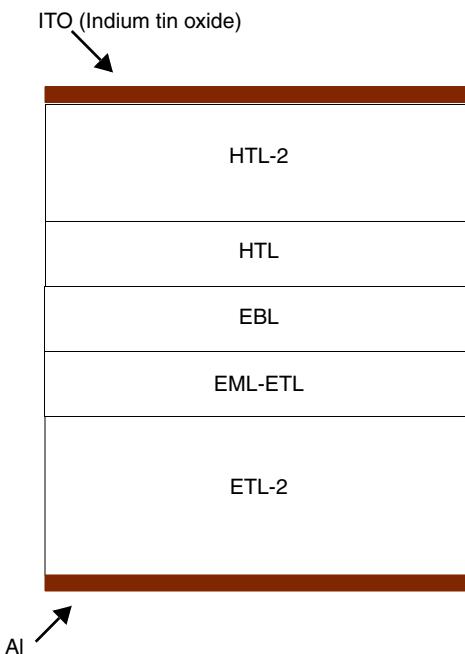


Figure 84 Typical OLED layers: electron transport layers (ETLs), hole transport layers (HTLs), electron blocking layer (EBL), and emission layer (EML)

28: Light-emitting Diodes

Modeling Organic Light-emitting Diodes

The OLED simulator is implemented in a similar fashion to the LED simulator (see [Simulating Organic Light-emitting Diode on page 676](#)).

Raytracing is used as the approximate optical solver (see [Raytracing on page 407](#) and [LED Raytracing on page 724](#)).

OLED Emissions

Other than the common organic transport models listed in [Chapter 13 on page 389](#), OLED simulation also requires an optical emission model. A simple organic optical emission model has been implemented that computes the optical spontaneous emission rate from the decay rate of singlet excitons.

The rate of photon generation is thereby given by the radiative decay of the singlet excitons, either directly or trap-assisted:

$$R_{\text{spon}} = \frac{n_{\text{se}} - n_{\text{se}}^{\text{eq}}}{\tau} + \frac{n_{\text{se}} - n_{\text{se}}^{\text{eq}}}{\tau_{\text{trap}}} \quad (780)$$

where:

- n_{se} is the density of singlet excitons.
- $n_{\text{se}}^{\text{eq}}$ is the equilibrium density of single excitons.
- τ and τ_{trap} are the lifetimes given in the `SingletExciton` section of the parameter file (see [Organic Material Parameters](#)).

Organic Material Parameters

In addition, you have to input parameters for the various organic models in the parameter file:

```
Region="HTL-2" { ...
    GaussianDOS {
        * A_e = sigmaDOS_e*sigmaDOS_e/2/kB/kB
        * B_e = E0_c/kB          (E0_c = E0 - Ec)
        * C_e = E0_c/sqrt(2)/sigmaDOS_e
        * Nc(T) = NLUMO/2 * exp(A_e/T/T - B_e/T) * erfc(sqrt(A_e)/T - C_e)

        * A_h = sigmaDOS_h*sigmaDOS_h/2/kB/kB
        * B_h = E0_v/kB          (E0_v = Ev - E0)
        * C_h = E0_v/sqrt(2)/sigmaDOS_h
        * Nv(T) = NHOMO/2 * exp(A_h/T/T - B_h/T) * erfc(sqrt(A_h)/T - C_h)

        * Gaussian DOS g(E)=Nt/sqrt(2*PI)/sigmaDOS
```

```

*
*exp(-(E-E0)*(E-E0)/2/sigmaDOS/sigmaDOS)
*Nt = NLUMO or NHOMO

N_LUMO      = 1e21      # [cm-3]
sigmaDOS_e  = 0.081    # [eV]
E0_c        = 0.1       # [eV]
N_HUMO      = 1e21      # [cm-3]
sigmaDOS_h  = 0.120    # [eV]
E0_v        = 0.1       # [eV]
}

PFMob
{ * mu = mu_0 * exp(-E0/KB/T) * exp( sqrt(F)*(beta/T - gamma) )
* where mu_0 is the low field mobility, beta and gamma are fitting
* parameters, E0 effective activation energy and F the driving force
* (electric field).
  beta_e     = 0.63      # [K (cm/V)^0.5]
  beta_h     = 0.63      # [K (cm/V)^0.5]
  E0_e       = 0.0
  E0_h       = 0.0
  gamma_e    = 0.0
  gamma_h    = 0.0
}

SingletExciton {
  gamma        = 0.25      # [1]
  l_diff       = 1e-8      # [cm^2/s]   # 10nm
  tau          = 1e-8      # [s]         # 10ns
  tau_trap     = 1e-8      # [s]
  ex_cXsection = 1e-14 , 1e-14 # [cm^2]
  vth          = 1e-1      # [cm/s]
  Eex          = 0.015    # [eV]
  gex          = 4          # [1]
  vth_car     = 1e1 , 1e1   # [cm/s]
}
}

```

References

- [1] W.-C. Ng and M. Pfeiffer, "Generalized Photon Recycling Theory for 2D and 3D LED Simulation in Sentaurus Device," in *6th International Conference on Numerical Simulation of Optoelectronic Devices (NUSOD-06)*, Singapore, pp. 127–128, September 2006.
- [2] W.-C. Ng and G. Létay, "A Generalized 2D and 3D White LED Device Simulator Integrating Photon Recycling and Luminescent Spectral Conversion Effects," in *Proceedings of SPIE, Light-Emitting Diodes: Research, Manufacturing, and Applications XI*, vol. 6486, February 2007.

This chapter describes various methods and boundary conditions used to compute the optical modes of lasers.

These methods include the finite-element method, the transfer matrix method, and the effective index method. Far-field derivation from the near-field pattern and methods for automatic mode searching are also described.

Overview

In a laser simulation, the optics problem is challenging. The refractive index distribution in a laser structure changes with temperature and carrier density (plasma effect). In Sentaurus Device, the refractive index is taken from a corresponding parameter file and wavelength dispersion of the refractive indices can also be taken into consideration (see [Refractive Index, Dispersion, and Optical Loss on page 699](#)). To handle such inhomogeneous refractive index geometries, the finite-element method was selected as the core method to discretize and solve the optics problem. In addition, there are other approximate methods such as the effective index method, transfer matrix method, and raytracing. These additional methods allow you to run the simulation faster, but with some loss of accuracy. They are described in detail in the following sections.

The resulting quantities of interest from the optics solution are the optical mode profiles, optical loss, and resonant wavelength. In a Fabry–Perot cavity for an edge-emitting laser, the wavelength is determined by the peak of the material gain from the electrical simulation. In other cases such as distributed feedback (DFB) lasers, distributed Bragg reflector (DBR) lasers, and VCSELs, the wavelength is determined by the shape and design of the cavity structure.

The optics problem can be iterated self-consistently with the electrical problem by Gummel iteration. Referring to [Simulating Single-Grid Edge-emitting Laser on page 658](#), the self-consistent coupling is activated by using the keyword Optics in the Plugin statement:

```
Solve {...  
quasistationary (...  
    Goal {name="p_Contact" voltage=1.6})  
{  
    Plugin(BreakOnFailure){  
        Coupled { Electron Hole Poisson QWeScatter QWhScatter  
                  PhotonRate }  
    }
```

29: Optical Mode Solver for Lasers

Finite-Element Formulation

```
Optics  
Wavelength  
}  
}  
}
```

If the coupling between the optics and electronics is weak, for example, in an isothermal simulation and at low-current injection conditions, it is not necessary to iterate the optics problem with the electrical problem since the optical eigenmode is not expected to change greatly. In such a case, remove the `Optics` keyword from the `Plugin` statement, and the optical eigenmode is computed only once at the beginning of the simulation.

There are two types of optical problem in laser simulations: the waveguide problem for edge-emitting lasers and the resonant cavity problem, for example, in VCSELs. In the waveguide problem, essentially, the Helmholtz equation is solved, while the cavity problem solves the wave equation directly. These two types of problem are discussed in the next sections.

Finite-Element Formulation

The finite-element method is a standard variational approach for solving the electromagnetic wave equations [1]. The Ritz procedure is used to evaluate the functional:

$$F(\Psi) = \frac{1}{2} \langle \Im \Psi, \Psi \rangle - \langle \Psi, f \rangle - \langle f, \Psi \rangle \quad (781)$$

where \Im is the linear operator and Ψ is a trial function. The governing differential equation is $\Im \Phi = f$. The wave equations in the waveguide and cavity problems are both of the form of the governing differential equation and, therefore, are well suited to the finite-element method.

For example, use the scalar Helmholtz equation for the waveguide problem to trace the formulation of the finite-element method. The Ritz procedure is applied to the scalar Helmholtz equation and gives the variational functional:

$$F(\Psi) = \frac{1}{2} \iint ((\partial_x \Psi)^2 + (\partial_y \Psi)^2 - k_0^2 \epsilon_r \Psi^2) d\Omega \quad (782)$$

The edge-emitting laser structure is discretized into finite elements, and the trial function Ψ is constructed by assuming weighted, linear basis functions on each edge of the finite element. These basis functions are commonly referred to as shape functions. In this case, the variables are the coefficients (weights) of the linear basis function on each edge. By the method of variation, the minimization of the functional $F(\Psi)$ gives the optimal solution, Ψ , for the finite-element discretization of the scalar Helmholtz equation. Therefore, the resolution of the discretization is important to determine the accuracy of the solution. As a general rule, 20 points per wavelength will provide an accurate solution for most structures.

To find the minimization of the functional $F(\Psi)$, take the first derivative of $F(\Psi)$ with respect to the weights of the linear basis function, $\{\Phi\}$, and set the derivative to zero. This yields a set of linear equations that can be cast into an algebraic, generalized, eigenvalue problem:

$$[A]\{\Phi\} = \varepsilon_{\text{eff}}[B]\{\Phi\} \quad (783)$$

with:

$$[A] = \sum_{\text{allElements}} \left(k_0^2 \varepsilon_r e \int_{\Omega^e} \{L\} \cdot \{L\}^T d\Omega^e - \int_{\Omega^e} \int \{\nabla_t L\} \cdot \{\nabla_t L\}^T d\Omega^e \right) \quad (784)$$

$$[B] = \sum_{\text{allElements}} \int_{\Omega^e} \int \{L\} \cdot \{L\}^T d\Omega^e \quad (785)$$

where $\{L\}$ are the linear shape functions. The relative dielectric constant ε_r remains constant across the element e , and the integration occurs over the area of the element, Ω^e .

The sparse matrices $[A]$ and $[B]$ are derived from the assembly process. They are complex symmetric but nonhermitian. As the system is nonhermitian, it causes the eigenvalues ε_{eff} to be complex. $\{\Phi\}$ is the column vector of the unknown weights. After the weights, $\{\Phi\}$, have been solved, they are substituted back into the linear shape functions to recover the electric fields on the nodes of the grid.

The Jacobi–Davidson QZ algorithm is applied [2][3] to solve the sparse matrix generalized eigenvalue problem. This is an iterative solver, so an initial guess for the eigenvalue is required. The better the initial guess, the faster the solution will be found. In addition, the numeric solver has also been parallelized.

Syntax of FE Scalar and FE Vectorial Optical Solvers

The finite-element (FE) scalar solver caters only to the waveguide problem, while the FE vectorial solver handles both waveguide and cavity problems. The choice of FE scalar or FE vectorial solvers is determined in the `Optics` statement. The default is the FE scalar solver for waveguide modes.

29: Optical Mode Solver for Lasers

Syntax of FE Scalar and FE Vectorial Optical Solvers

FE Scalar Solver

The `FEScalar` solver for the waveguide problem is activated in the `Physics-Laser-Optics` statement in the command file:

```
Physics {...  
  Laser (...  
    Optics(  
      FEScalar(EquationType = Waveguide  
        Symmetry = Symmetric  
        LasingWavelength = 800      # [nm]  
        TargetEffectiveIndex = 3.4  # initial guess  
        TargetLoss = 10.0          # initial guess [1/cm]  
        Polarization = TE  
        Boundary = "Type1"  
        ModeNumber = 1  
      )  
    )  
  )  
}
```

This example shows how to activate the FE scalar solver to solve for one waveguide mode. As discussed in [Finite-Element Formulation on page 762](#), an iterative method is used to solve for the modes. Therefore, specifying a good `TargetEffectiveIndex` is important to speed up the computation of the solution. For multimodes, increase `ModeNumber` and specify multiple entries for `TargetEffectiveIndex`, `TargetLoss`, and so on (see [Specifying Multiple Entries for Parameters in FEScalar and FEVectorial on page 767](#)). Only Cartesian coordinates and waveguide modes are associated with the FE scalar solver. For scalar cavity solvers for VCSELs, refer to the transfer matrix method and effective index method.

In Sentaurus Device, there is an option to run only the optical solver without activating the laser simulation. This is called the optics stand-alone option (see [Optics Stand-alone Option on page 795](#)). In this case, you must specify the keyword `Boundary` in the `FEScalar` statement if `Symmetric` or `Periodic` is chosen. In a laser simulation, the default values of `Boundary` listed in [Table 111 on page 770](#) are used unless you select the boundary types explicitly. A detailed discussion of the `Boundary` keyword is in [Boundary Conditions and Symmetry for Optical Solvers on page 769](#).

[Table 189 on page 1154](#) describes all of the possible arguments that can be used inside the `FEScalar` statement.

There are three types of symmetry entry:

- `Symmetric` is symmetry about the y-axis at $x = 0$.
- `Nonsymmetric` is nonsymmetry.
- `Periodic` means the left and right boundaries of the Neumann type.

The default boundary conditions for the different symmetry types are listed in [Table 111 on page 770](#). The `LasingWavelength` entered is solely for the initial computation of the optical mode and the way the lasing wavelength is computed is discussed in [Waveguide Optical Modes and Fabry–Perot Cavity on page 703](#). `TargetEffectiveIndex` is the initial guess for the eigenvalue of the waveguide problem and is a necessary input to ensure that the required mode is computed.

FE Vectorial Solver

The FE vectorial solver handles both waveguide and cavity-type problems. The syntax of both problems is described separately.

FE Vectorial Solver for Waveguides

The syntax for using the vectorial FE solver for waveguides is:

```
Physics {...  
    Laser (...  
        Optics(  
            FEVectorial(EquationType = Waveguide  
                Symmetry = Symmetric  
                Coordinates = Cartesian  
                LasingWavelength = 800      # [nm]  
                TargetEffectiveIndex = 3.4 # initial guess  
                TargetLoss = 10.0          # initial guess [1/cm]  
                Boundary = "Type1"  
                ModeNumber = 1  
            )  
        )  
    )  
}
```

The argument list for `FEVectorial` statement for the waveguide problem is similar to that of the `FEScalar` statement.

29: Optical Mode Solver for Lasers

Syntax of FE Scalar and FE Vectorial Optical Solvers

FE Vectorial Solver for VCSEL Cavity

The syntax for using the FE vectorial solver for a VCSEL cavity is:

```
Physics {...  
    Laser (...  
        Optics(  
            FEVectorial(EquationType = Cavity  
                Coordinates = Cylindrical  
                TargetWavelength = 777    # initial guess [nm]  
                TargetLifeTime = 1.4      # initial guess [ps]  
                AzimuthalExpansion = 1   # cylindrical harmonic order  
                ModeNumber = 1  
            )  
        )  
        VCSEL()    # specify this is a VCSEL simulation  
    )  
}
```

The syntax for the VCSEL cavity problem is very different from that of the waveguide problem. The keyword `VCSEL` must be included in the `Laser` statement to indicate that this is a VCSEL simulation. An initial guess for the resonant wavelength must be specified by `TargetWavelength`.

Cylindrical symmetry has been specified by `Coordinates=Cylindrical`, so the modes contain the angular dependence of $e^{im\phi}$. The cylindrical harmonic order, m , can be changed by the keyword `AzimuthalExpansion`.

NOTE Unless stated specifically, the argument should apply to both cavity and waveguide problems.

[Table 190 on page 1155](#) describes the arguments that can be used with the `FEVectorial` keyword. [Table 108](#) and [Table 109 on page 767](#) group the keywords for the waveguide and cavity problems, respectively.

Table 108 Dependencies of keywords for `EquationType=Waveguide` in `FEVectorial` optical solver

Keyword	Dependencies				
EquationType	Waveguide				
Coordinates	Not valid				
Symmetry	Symmetric		NonSymmetric	Periodic	
Boundary	Type1	Type2	Not valid	Type1	Type2
TargetWavelength	Not valid				

Table 108 Dependencies of keywords for EquationType=Waveguide in FEVectorial optical solver

Keyword	Dependencies
TargetLifetime	Not valid
LongitudinalWavevector	Not valid
AzimuthalExpansion	Not valid
LasingWavelength	<float>
TargetEffectiveIndex	<float>
TargetLoss	<float>

Table 109 Dependencies of keywords for EquationType=Cavity in FEVectorial optical solver

Keyword	Dependencies			
EquationType	Cavity			
Coordinates	Cartesian		Cylindrical	
Symmetry	Symmetric	NonSymmetric	Not valid	
Boundary	Type1	Type2		
TargetWavelength	<float>			
TargetLifetime	<float>			
LongitudinalWavevector	<float>		Not valid	
AzimuthalExpansion	Not valid		<int>	
LasingWavelength	Not valid			
TargetEffectiveIndex	Not valid			
TargetLoss	Not valid			

Specifying Multiple Entries for Parameters in FEScalar and FEVectorial

When multimodes are needed, `ModeNumber` is used to specify the number of modes required. In this case, you may want to specify different initial guesses for the effective index (for waveguide modes), resonant wavelength (for cavity modes), and so on. This is accomplished by extending the syntax for these initial guess parameters. The following is an example for waveguide modes and one for VCSEL cavity modes.

29: Optical Mode Solver for Lasers

Syntax of FE Scalar and FE Vectorial Optical Solvers

Multiple Waveguide Modes

The syntax extension of `FEScalar` for multiple waveguide modes is shown here. `FEVectorial` for multiple waveguide modes has the same syntax extension. The boundary type is specified explicitly in this example. If the `Boundary` keyword is omitted, the default values in [Table 111 on page 770](#) are taken:

```
Physics {...
    Laser {...}
    Optics(
        FEScalar(EquationType = Waveguide
            Symmetry = Symmetric
            LasingWavelength = 780
            TargetEffectiveIndex = (3.54 3.47 3.5 3.33 3.4)
            TargetLoss = (5.0 6.0 8.0 7.0 10.0)
            Polarization = (TE TE TM TE TM)
            ModeNumber = 5
            Boundary = ("Type1" "Type2" "Type1" "Type1" "Type2")
        )
    )
}
```

Multiple Cavity Modes

The syntax extension of `FEVectorial` for multiple cavity modes is:

```
Physics {...
    Laser {...}
    Optics(
        FEVectorial(EquationType = Cavity
            Coordinates = Cylindrical
            TargetWavelength = (777 762 760 753 751)
            TargetLifeTime = (1.5 1.3 1.22 1.16 1.1)
            AzimuthalExpansion = (1 0 0 2 2)
            ModeNumber = 5
        )
    )
    VCSEL()
}
```

Boundary Conditions and Symmetry for Optical Solvers

There are three main types of boundary condition for the optical problem:

- Dirichlet boundary condition
- Neumann boundary condition
- Radiative boundary condition

These boundary conditions can be controlled to some extent with the `Symmetry` keyword in the `FEScalar` and `FEVectorial` statements. In the optical solvers of Sentaurus Device, the external boundaries of the optical simulation space are assumed (by default) to satisfy the Dirichlet boundary condition, that is, the optical fields on this outer boundary are set to zero. By specifying different types of symmetry, the Neumann boundary condition can be imposed on different external boundaries.

The boundary conditions for raytracing are treated differently and are discussed in [Boundary Condition for Raytracing on page 422](#).

Table 110 summarizes the symmetry types (appearing in the `FEScalar` and `FEVectorial` statements) and lists the associated boundary conditions.

Table 110 Symmetry types in `FEScalar` and `FEVectorial` statements and their associated optical boundary conditions

Symmetry type	Boundary condition
NonSymmetric	Dirichlet boundary condition on all external boundaries, that is, the optical field is set to zero at the boundaries.
Periodic	Neumann or Dirichlet boundary conditions on all vertical boundaries, and Dirichlet boundary condition on all horizontal boundaries.
Symmetric	Neumann or Dirichlet boundary condition for even or odd on the symmetry axis, and Dirichlet boundary condition elsewhere. The symmetry axis is defined as the y-axis at $x = 0$.

If `Symmetry=Symmetric` and the optics stand-alone option is used, the argument keyword `Boundary` must be specified. In laser simulations, `Boundary` is chosen automatically unless explicitly specified. The argument `Boundary` dictates the boundary condition at the symmetry y-axis and is specific only for Cartesian coordinates because odd and even modes require different types of boundary condition at the symmetry y-axis. A summary of the default boundary conditions used in different cases is presented in [Table 111 on page 770](#). The following examples explain the different cases.

29: Optical Mode Solver for Lasers

Boundary Conditions and Symmetry for Optical Solvers

Table 111 Default boundary conditions for the optical solvers

Case	Periodic	Symmetric	NonSymmetric
FEScalar	Boundary = "Type1" for first n/2 modes and Boundary = "Type2" for the rest of the modes.		Boundary not used.
FEVectorial, Waveguide	Boundary = "Type2" for first n/2 modes and Boundary = "Type1" for the rest of the modes.		
FEVectorial, Cavity	Boundary chosen automatically.		

NOTE The boundary condition for every mode can be explicitly specified by the keyword `Boundary = ("Type2" "Type1" ...)`.

Symmetric FEScalar Waveguide Mode in Cartesian Coordinates

In this example, the `FEScalar` optical mode solver is applied to a symmetric edge-emitting laser to obtain the even and odd modes. The top row of Figure 85 shows the even modes. They are calculated when the keyword `Boundary = "Type1"` is set. The bottom row shows the odd modes. They are computed when the keyword `Boundary = "Type2"` is set. The fundamental mode is obtained if the keyword `Boundary = "Type1"` is set while the first-order mode is calculated for `Boundary = "Type2"`.

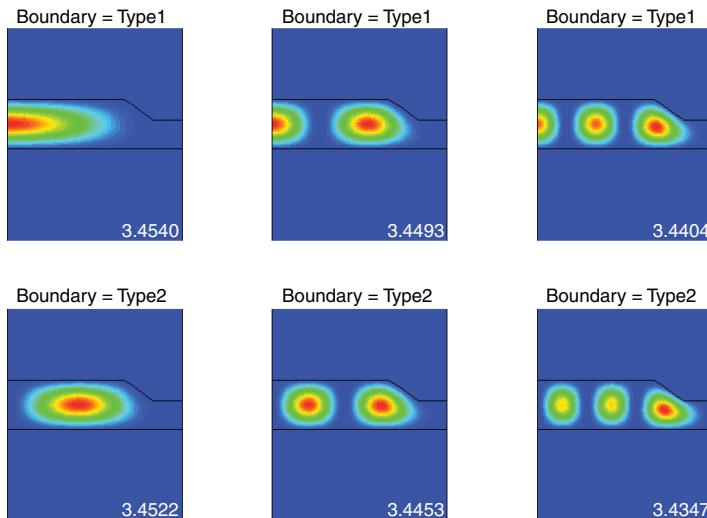


Figure 85 Scalar mode patterns of a symmetric edge-emitter structure

Symmetric FEVectorial Waveguide Modes in Cartesian Coordinates

In this example, the FEVectorial optical mode solver is applied to a symmetric edge-emitting laser to find the horizontally and vertically polarized modes. The top row of [Figure 86](#) shows the calculated modes when the keyword `Boundary = "Type1"` is set. The bottom row shows the modes that are calculated when the argument `Boundary = "Type2"` is set. The horizontally polarized fundamental mode is obtained for `Boundary = "Type2"`.

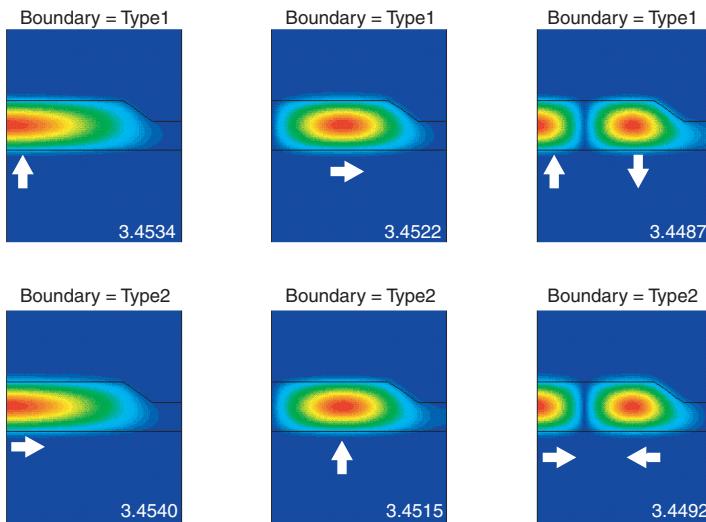


Figure 86 Vectorial mode patterns of a symmetric edge-emitter structure

Symmetric FEVectorial VCSEL Cavity Modes in Cartesian Coordinates

In addition to cylindrical symmetry, the FEVectorial optical mode solver can also be applied to a symmetric VCSEL in Cartesian coordinates to find the modes that are polarized in-plane or perpendicular to the plane.

The top row of [Figure 87 on page 772](#) shows the modes when the keyword `Boundary = "Type1"` is set. The bottom row shows the modes that are computed when the keyword `Boundary = "Type2"` is set.

The in-plane polarized fundamental mode is obtained for `Boundary = "Type2"`, while the perpendicularly polarized fundamental mode is computed for `Boundary = "Type1"`.

29: Optical Mode Solver for Lasers

Boundary Conditions and Symmetry for Optical Solvers

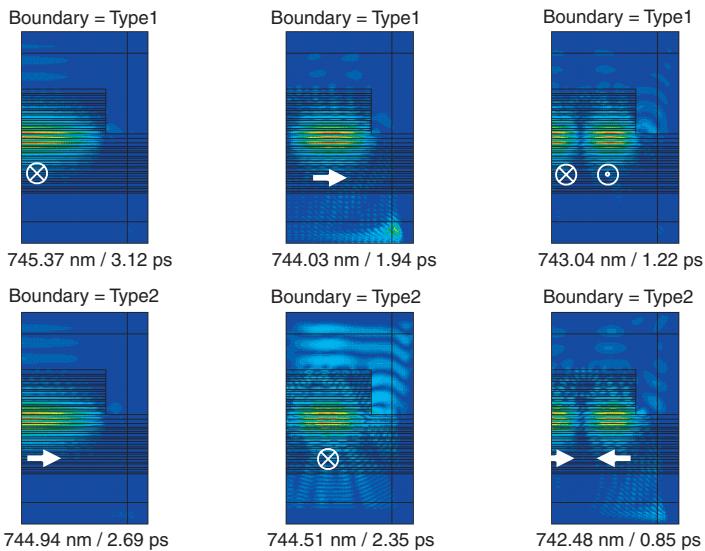


Figure 87 Vectorial mode patterns of a VCSEL in Cartesian coordinates

Symmetric FEVectorial VCSEL Cavity Modes in Cylindrical Coordinates

The argument `AzimuthalExpansion` in the `FEVectorial` optical mode solver is used to calculate different types of mode in a cylindrical VCSEL. The left column of [Figure 88 on page 773](#) shows the computed modes for the argument `AzimuthalExpansion=0`. The middle column shows the modes that are calculated when `AzimuthalExpansion=1` is set. The right column shows the mode pattern for `AzimuthalExpansion=2`. For example, the fundamental mode HE11 is obtained by setting the argument `AzimuthalExpansion=1`.

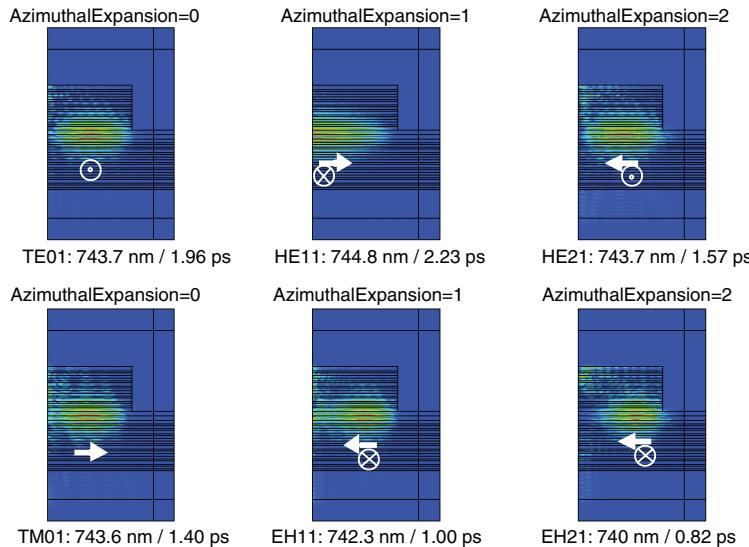


Figure 88 Vectorial mode patterns of a VCSEL in cylindrical coordinates

Perfectly Matched Layers

Apart from Dirichlet and Neumann boundary conditions, Sentaurus Device can simulate the radiative boundary condition artificially using the concept of a perfectly matched layer (PML).

The PML in Sentaurus Device is implemented by using a tensorial permeability quantity [4], which can be interpreted as a uniaxial anisotropic medium. This can be proven to be the same as coordinate stretching for the curl, divergence, and gradient operators [5]. (Further information is available from the literature: the original PML work [6], PML interpreted as a coordinate-stretching concept [5], and the generalization of the PML concept to various coordinate systems and general anisotropic and dispersive media [7][8].) A mathematical presentation of the PML is beyond the scope of this manual. Therefore, only the physical concept behind the method is discussed.

NOTE Perfectly matched layers should only be added when using the vectorial (FEVectorial) optical solver.

The Dirichlet boundary condition is assumed at the outer boundaries of the simulation space. In many cases, the optical field may radiate outwards, for example, waves from the lasing region leak into the substrate because the substrate has about the same refractive index as the guiding layers. In this case, the solution obtained using the Dirichlet boundary condition will include reflections of these radiative waves from the boundaries, which are nonphysical. To solve this problem, the boundaries of the structure can be coated with PMLs to reduce and

29: Optical Mode Solver for Lasers

Perfectly Matched Layers

eliminate unwanted reflections from the Dirichlet boundary condition in a truncated simulation space. This enables the artificial simulation of the radiative boundary condition.

The structure is terminated by an inner boundary Γ_i (see [Figure 89](#)), and the PML is placed between the inner boundary Γ_i and outer boundary Γ_o . A gradually increasing loss is introduced in the PML from the inner boundary Γ_i towards the outer boundary Γ_o . Therefore, upon first impact of the wave into the PML at the inner boundary Γ_i , negligible reflection occurs. As the wave propagates deeper into the PML, it is absorbed more and more. Any reflected waves within the PML also suffer from absorption. Ultimately, it appears that the propagating wave in the PML is totally absorbed and, therefore, this is how the PML simulates the radiative boundary condition. The loss profile in the PML has been set automatically in Sentaurus Device.

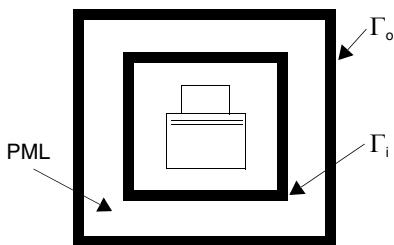


Figure 89 PML coating the structure

PMLs are indicated by special keywords appended to the beginning of the region names when the optical device is drawn. The region names of PMLs at the top, bottom, left, and right of the structure start with TPML, BPML, LPML, and RPML, respectively, as shown in [Figure 90](#).

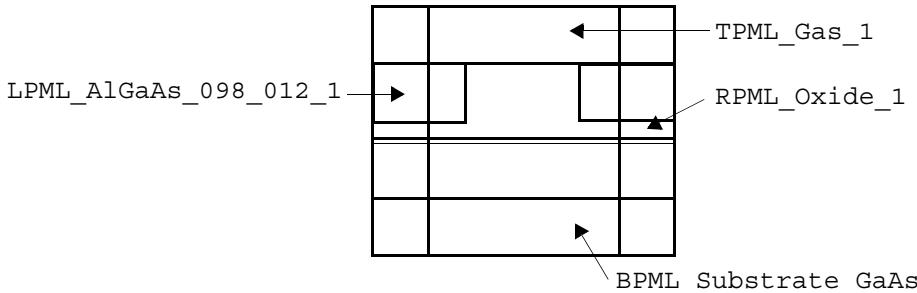


Figure 90 Naming of PML regions

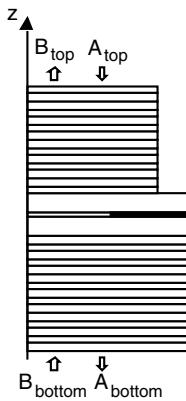
This naming convention is required to impose the correct boundary condition in the optical solver. These special PML regions are declared in the same way as other regions in the command file of Sentaurus Device. A Tcl-based script can be used to add automatically PML regions to the outer boundaries of the structure. This script can be obtained from the Synopsys Technical Support Center.

Transfer Matrix Method for VCSELs

The 1D transfer matrix method (TMM) is a simplified scalar solver for the VCSEL cavity problem. It computes the spatial optical intensity and the corresponding characteristics of the fundamental mode in a cylindrically symmetric VCSEL structure. The scalar wave equation in the axial direction is:

$$\left(\frac{\partial^2}{\partial z^2} + k_z^2 \right) \phi_z = 0 \quad (786)$$

where ϕ_z denotes the wave component in the axial direction. The TMM is applied at the symmetry axis in the vertical direction (see [Figure 91](#)).



[Figure 91](#) Transfer matrix method applied to a symmetric VCSEL structure

The solution to the axial wave equation in each layer i can be expressed as the sum of a forward-propagating and backward-propagating wave:

$$\phi_z = A(z_i) e^{(ik_{zi}z)} + B(z_i) e^{(-ik_{zi}z)} \quad (787)$$

The TMM relates the two waves at the interfaces i and $i + 1$ by:

$$\begin{bmatrix} A(z_i) \\ B(z_i) \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix}_i \cdot \begin{bmatrix} A(z_{i+1}) \\ B(z_{i+1}) \end{bmatrix} \quad (788)$$

The transfer matrix for an index step $n_1 \rightarrow n_2$ is:

$$T_{1 \rightarrow 2} = \frac{1}{t_{12}} \begin{bmatrix} 1 & r_{12} \\ r_{12} & 1 \end{bmatrix} \quad (789)$$

29: Optical Mode Solver for Lasers

Transfer Matrix Method for VCSELs

$$\text{with } r_{12} = -r_{21} = \frac{n_1 - n_2}{n_1 + n_2} \text{ and } t_{12} = t_{21} = \frac{2n_1}{n_1 + n_2}.$$

For a homogeneous medium of length d , the transfer matrix becomes:

$$T_d = \begin{bmatrix} e^{ikd} & 0 \\ 0 & e^{-ikd} \end{bmatrix} \quad (790)$$

With these basic transfer matrix blocks, a final transfer matrix can be set up that relates the forward-propagating and backward-propagating waves at the top ($A_{\text{top}}, B_{\text{top}}$) of the device to the wave at the bottom ($A_{\text{bottom}}, B_{\text{bottom}}$) of the device:

$$\begin{bmatrix} B_{\text{top}} \\ A_{\text{top}} \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix}_0 \cdot \dots \cdot \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix}_{2n+1} \cdot \begin{bmatrix} B_{\text{bottom}} \\ A_{\text{bottom}} \end{bmatrix} = \begin{bmatrix} T_{11, \text{tot}} & T_{12, \text{tot}} \\ T_{21, \text{tot}} & T_{22, \text{tot}} \end{bmatrix} \cdot \begin{bmatrix} B_{\text{bottom}} \\ A_{\text{bottom}} \end{bmatrix} \quad (791)$$

where n denotes the number of layers of the structure. Resonance is achieved only if the outward-propagating waves are established at the top and bottom of the device ($A_{\text{top}} = 0$ and $B_{\text{bottom}} = 0$). This condition is found by varying the frequency ω and the gain in the quantum wells. The corresponding characteristics of this instance are the resonant wavelength and quantum well gain. At sustained resonance, the threshold gain in the quantum wells must balance the radiation losses from the device.

Since the 1D TMM only provides the change of field in the axial direction, a Gaussian variation is assumed for the optical intensity in the transverse direction:

$$\text{Gauss}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \quad (792)$$

where x is the distance in the transverse direction from the symmetry axis and σ is the width of the Gaussian shape.

The activation syntax for the TMM for VCSEL cavity problems in the command file is:

```
Physics {...  
    Laser (...  
        Optics(  
            TMM1D(SigmaGauss=4.0      # width of Gaussian, [micrometer]  
            TargetWavelength=800      # initial guess [nm]  
        )  
    )  
    VCSEL(                      # specify this is a VCSEL simulation  
    )  
}
```

The solution of the resonant wavelength is solved iteratively, so you must specify an initial guess in `TargetWavelength`. The default and only symmetry type supported by `TMM1D` is `Symmetric`; only the fundamental mode is computed. The rest of the entries in the `Physics` and `Laser` statements are similar to that in [Simulating Single-Grid Edge-emitting Laser on page 658](#).

[Table 197 on page 1160](#) lists the arguments of `TMM1D`.

Effective Index Method for VCSELs

The effective index method (EIM) is a fast scalar solver and well suited to computing fairly accurate resonant wavelength and optical intensity for index-guided VCSELs. However, the EIM cannot compute the scattering losses accurately for very small aperture (less than 2 μm radius) VCSELs. Nevertheless, the EIM is an option in Sentaurus Device to cater to the rapid design of index-guided VCSELs, including oxide-confined VCSELs.

[Figure 92](#) shows a typical VCSEL geometry suitable for the EIM. The structure is divided into two regions: the core and the cladding. Within each core and cladding region, multiple homogeneous layers of semiconductor are allowed.

NOTE In Sentaurus Device, the EIM is only applicable to the VCSEL cavity problem, *not* the waveguide problem.

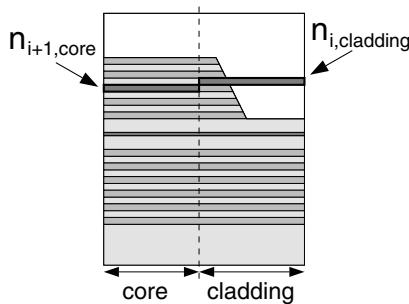


Figure 92 VCSEL partitioned into the core and cladding regions for the EIM

Formulation of Effective Index Method

The EIM solves the scalar Helmholtz equation:

$$\nabla^2 \Psi + k^2 \Psi = 0 \quad (793)$$

by assuming that the wavefunction Ψ is separable, that is:

$$\Psi = \phi_t(x, y) \cdot \phi_z(z) \quad (794)$$

where the subscripts t and z refer to the transverse and z components. Substituting [Eq. 794](#) into [Eq. 793](#) gives two independent equations:

The axial wave equation is:

$$\left[\frac{\partial^2}{\partial z^2} + k_z^2 \right] \cdot \phi_z = 0 \quad (795)$$

The transverse wave equation is:

$$\left[\nabla_t^2 + \left[n(r) \cdot \frac{2\pi}{\lambda_0} \right]^2 - \beta^2 \right] \cdot \phi_t = 0 \quad (796)$$

The transverse and axial wave equations are coupled using the dispersion relation:

$$k^2 = k_t^2 + k_z^2 = n^2 \cdot k_0^2 \quad (797)$$

where the free space wavenumber is $k_0 = \frac{2\pi}{\lambda_0}$.

The solution strategy of the EIM for VCSELs is best illustrated by the flowchart in [Figure 93](#) on page [779](#).

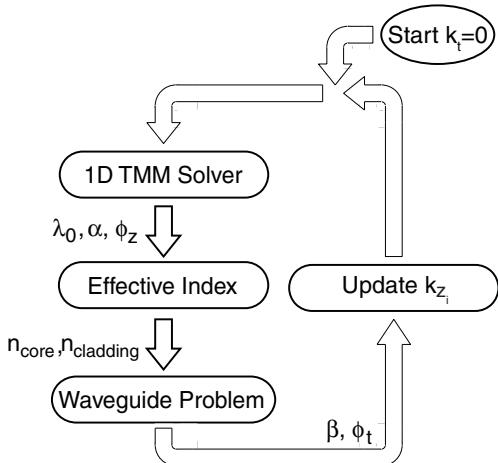


Figure 93 Solving the axial and transverse wave equations self-consistently in the EIM

First, the axial wave equation is solved by the transfer matrix method. The required output is the resonant wavelength λ_0 , the net cavity loss α , and the z-resonant field $\phi_z(z)$.

Next, $\phi_z(z)$ is used to compute the effective indices in the core (n_{core}) and the cladding (n_{cladding}) regions by the following averaging relation:

$$n_{\text{core,cladding}}^2 = \frac{\sum_i \int_{z_i}^{z_{i+1}} n_{i, \text{core/cladding}}^2 \cdot |\phi_z|^2 dz}{\int |\phi_z|^2 dz} \quad (798)$$

The refractive index of each layer i is weighted by the z-resonant optical intensity, and the effective index can be viewed as an average of these weighted refractive indices.

n_{core} and n_{cladding} are subsequently used in the transverse wave equation to formulate an optical fiber or waveguide-type problem.

Solving the transverse wave equation yields the propagation constant β , which is then used to update k_{z_i} of the axial wave equation using the relations:

$$\left(n_{\text{core}} \cdot \frac{2\pi}{\lambda_0}\right)^2 = k_t^2 + \beta^2 \quad (799)$$

and:

$$k_{z_i}^2 = \left(n_i \cdot \frac{2\pi}{\lambda_0}\right)^2 - k_t^2 \quad (800)$$

29: Optical Mode Solver for Lasers

Effective Index Method for VCSELs

This is performed so that the concept of phase-matching is enforced at the tangential boundaries of all the layers. The entire process is iterated until convergence is achieved.

Transverse Mode Pattern of VCSELs

The transverse wave equation (Eq. 796) in the EIM can be solved in two different coordinate systems: Cartesian and cylindrical coordinates. The core and cladding effective indices form a symmetric three-layer waveguide problem. In Cartesian coordinates, this is equivalent to the classic step-index planar waveguide problem. In cylindrical coordinates, this becomes an optical fiber problem. Both problems can be solved by a semianalytic approach, and the range of effective index in this waveguide problem is:

$$n_{\text{cladding}} < n_{\text{eff}} < n_{\text{core}} \quad (801)$$

where the propagation constant $\beta = n_{\text{eff}} \cdot k_0$. This ensures that the transverse mode is a guided mode.

The step-index planar waveguide in Cartesian coordinates is an approximation to solving the transverse mode profile in square aperture VCSELs. It is assumed that the square aperture VCSEL is symmetric to a plane, as shown in Figure 94. The core and cladding indices form the step-index layers of the waveguide, and the TE modal field component, $\phi_t(x, y) = \phi_y(x)$, varies as $\sin(x)$ or $\cos(x)$ in the core region and as $\exp(-|x|)$ in the cladding region.

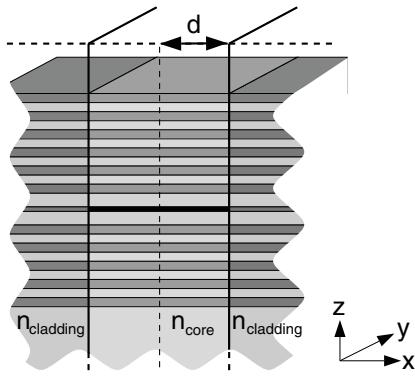


Figure 94 Transverse mode in square aperture VCSEL is treated as a step-index planar waveguide problem

In a circular aperture VCSEL (see [Figure 95](#)), the optical fiber problem is solved. The modal field components, $\phi_t(r, \varphi)$, vary as $J_{m-1}(r)$ (Bessel function) in the core region and as $K_{m-1}(r)$ (modified Bessel function) in the cladding region. The parameter m denotes the cylindrical harmonic order, $e^{im\varphi}$. The modes in an optical fiber are commonly classified as HE_{mn} , EH_{mn} , TE_{0n} , and TM_{0n} , and this convention is followed in the command file syntax. Generally, the fundamental mode is HE_{11} , followed by the higher order modes TE_{01} , TM_{01} , and so on.

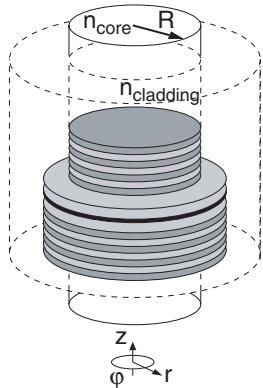


Figure 95 Transverse mode in circular aperture VCSEL is treated as an optical fiber problem

Syntax for Effective Index Method

The EIM solver can be activated by specifying the keyword `EffectiveIndex` in the `Physics-Laser-Optics` part of the command file.

Cylindrical Modes

```
Physics {...  
  Laser (...  
    Optics (  
      EffectiveIndex (  
        TargetWavelength = (780.0 750.0 775.0 770.0) # [nm]  
        CoreWidth = 4.0 # [micrometer]  
        ModeType = (HEmn EHmn TE0n TM0n) # choose mode type  
        mModeIndex = (1 1 0 0) # m in HEmn, EHmn  
        nModeIndex = (1 2 2 1) # n in HEmn, EHmn, TE0n, TM0n  
        Coordinates = Cylindrical  
        Absorption(ODB)  
        DiffractionLoss = (5.0 8.0 6.0 7.0) # [1/cm]  
        ModeNumber = 4 # solve for 4 modes  
      )  
    )
```

29: Optical Mode Solver for Lasers

Effective Index Method for VCSELs

```
        VCSEL()
    )
}
```

Cartesian Modes

```
Physics {...}
Laser (... {
    Optics (
        EffectiveIndex (
            TargetWavelength = (780.0 750.0 775.0 770.0)    # [nm]
            CoreWidth = 4.0                                # [micrometer]
            ModeType = (TE0n TE0n TE0n TE0n) # only TE0n allowed for Cartesian
            nModeIndex = (1 2 3 4)                      # n in TE0n
            Coordinates = Cartesian
            Absorption(ODB)
            DiffractionLoss = (5.0 8.0 6.0 7.0) # [1/cm]
            ModeNumber = 4                            # solve for 4 modes
        )
    )
    VCSEL()
)
}
```

This example shows the activation of the EIM for multiple cylindrical and Cartesian modes. The more notable aspects of the syntax are:

- The `CoreWidth` is the radius of the circular aperture in `Cylindrical` coordinates or the half-length of the square aperture in `Cartesian` coordinates.
- The keywords `HEmn`, `EHmn`, `TE0n`, and `TM0n` refer to the common optical fiber modes in the cylindrical modes. For the Cartesian modes, only `TE0n` modes are handled.
- The keywords `ModeType`, `mModeIndex`, and `nModeIndex` are used to specify the required modes.
- The keyword `VCSEL` must be included in the `Laser` statement to specify that this is a VCSEL simulation.
- The number of modes to solve is four in this example, and the parameters corresponding to each mode are specified as shown.

[Table 187 on page 1152](#) lists the full range of keywords for the EIM solver.

A sample output from the EIM solver is shown in [Figure 96 on page 783](#). The peak of the z-resonant mode aligns with the active quantum well region. Two vectorial transverse modes, `HE11` and `HE21`, computed from the optical fiber problem are also shown.

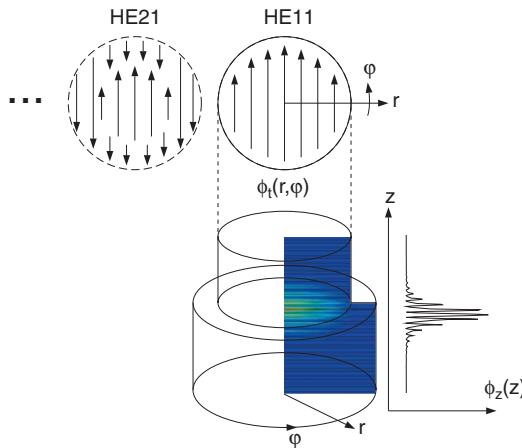


Figure 96 Transverse and longitudinal resonant modes of cylindrically symmetric VCSEL computed by EIM

Saving and Loading Optical Modes

The calculation of the optical modes of a laser can be time-consuming especially for large geometries. To save simulation time, transverse optical mode patterns can be saved and loaded from data files. However, when the optical modes are loaded from a file, the optical modes are assumed to be constant throughout the simulation. As a result, self-consistent iteration between the optics and electronics is not possible in this case.

Saving Optical Modes on Optical or Electrical Mesh

The optical field can be saved on the grid of the optical device (referred to as the optical mesh) in a dual-grid simulation by using the keyword `SaveOptField`. To understand more about dual-grid simulation, see [Simulating Dual-Grid Edge-emitting Laser on page 662](#). If the optical mesh is not found, the keyword `SaveOptField` saves the optical field to the only grid that is available – the electrical mesh as it is in [Simulating Single-Grid Edge-emitting Laser on page 658](#). The activation keyword is `SaveOptField` in the `File` and `Solve-quasistationary` sections of the command file:

```
File {...  
    SaveOptField = "laserfield"  
}  
...  
Solve {...  
    quasistationary (  
        SaveOptField { range=(0,1) intervals=3 }  
    )  
}
```

29: Optical Mode Solver for Lasers

Saving and Loading Optical Modes

```
Goal({name="p_Contact" voltage=1.8})  
{...}  
}
```

Refer to [Simulating Single-Grid Edge-emitting Laser on page 658](#) to see where this syntax is inserted in the command file. The `SaveOptField` has the same format as the `OptFarField` (see [Far Field on page 786](#)) and `GainPlot` (see [Plotting Gain and Power Spectra on page 855](#)). In this way, you can track the evolution of the optical field as the bias increases. With regard to the syntax and its output:

- In the `File` section, saving the optical field is activated by the keyword `SaveOptField`. The value assigned to it, "laserfield" in this case, becomes the base name for the output files of the optical fields.
- In the `Solve-quasistationary` section, the argument `range=(0,1)` in the `SaveOptField` keyword is mapped to the initial and final bias conditions. In this example, the initial and final (goal) `p_Contact` voltages are 0 V and 1.8 V, respectively. The number of `intervals=3`, which gives a total of four (= 3+1) optical field saved at 0 V, 0.6 V, 1.2 V, and 1.8 V. In general, specifying `intervals=n` will produce $(n+1)$ sets of optical field files.
- The output files in this example are:
 - `laserfield_000000_mode0_des.dat`
 - `laserfield_000001_mode0_des.dat`
 - `lasefield_000002_mode0_des.dat`
 - `lasefield_000003_mode0_des.dat`

Using a vectorial optical solver, each file contains four datasets: the optical field intensity, real and imaginary parts of the vectorial optical fields, and polarization. For the scalar optical solver `FEScalar`, only the optical field intensity and polarization are saved.

- For an LED simulation, only the intensity file is produced because the LED raytracing contains no vectorial or phase information.

Loading Optical Modes

Sentaurus Device can read optical modes that have been computed separately. The activating keywords are `OptField<n>` in the `File` section and `OpticsFromFile` in the `Physics` section. `OptField<n>` specifies the file from which optical intensity and polarization are loaded. Optical mode properties are specified in the `OpticsFromFile` section. Its keywords are summarized in [Table 193 on page 1157](#).

The following example shows the syntaxes for loading optical fields for an edge-emitting laser and a VCSEL.

Loading Optical Fields for Edge-emitting Laser

TargetEffectiveIndex and TargetLoss specify the real and imaginary part of the complex propagation constant of the mode. LasingWavelength defines the lasing wavelength of the laser.

```
File {...  
    OptField0 = "field_mode0_des.tdr"  
    OptField1 = "field_mode1_des.tdr"  
}  
...  
Physics {...  
    Laser (...  
        Optics (...  
            OpticsFromFile (  
                TargetEffectiveIndex = (3.45 3.44) # [1]  
                TargetLoss = (5.3510e-7 1.0400e-6) # [1]  
                LasingWavelength = 850.0 # [nm]  
                ModeNumber = 2  
            )  
        )  
    )  
}
```

Loading Optical Fields for VCSEL

TargetWavelength and TargetLoss specify the lasing wavelength and total decay rate. OutcouplingLoss defines the decay rate through the top mirror.

```
File {...  
    OptField0 = "field_mode0_des.dat"  
}  
...  
Physics {...  
    Laser (...  
        Optics (...  
            OpticsFromFile (  
                TargetWavelength = 750.19 # [nm]  
                TargetLoss = 0.0821 # [1/ps]  
                OutcouplingLoss = 0.0679 # [1/ps]  
                ModeNumber = 1  
            )  
        )  
    )  
    VCSEL()  
    OpticalConfinementFactor = 0.0338  
}  
}
```

29: Optical Mode Solver for Lasers

Far Field

With regard to the syntax:

- The number of optical field files specified must match the number of modes specified by ModeNumber.
- The number of target values must match the number of modes specified by ModeNumber.
- Up to ten optical modes can be selectively load.

NOTE When optical modes are read from files, self-consistent simulation between the optics and electronics is not possible.

The optical mode that is read into Sentaurus Device must strictly be on the same mesh as the optics would be computed. If the optical mode has been saved on another mesh (for example, in an optics stand-alone simulation with a different mesh), it must be interpolated to the corresponding mesh. This is accomplished by using DIP. An example that shows how to interpolate the optical intensity and polarization is:

```
set A [dip_mesh2D -args NULL fine_msh.grd field_des.dat]
set B [dip_mesh2D -args NULL coarse_msh.grd 0]
$B importDatasets $A new OpticalIntensityMode0
$B importDatasets $A new OpticalPolarizationAngleMode0
$B writeDatasets new_field_des.dat 0
```

These commands can be saved in a file, for example, `interpolate.cmd`, and the DIP interpolation routine can be called by `dipsh interpolate.cmd`.

Far Field

The far field is important to determine the beam divergence of the laser diode emission. From antenna theory, the far field is defined for observation distance:

$$r \geq \frac{2D^2}{\lambda} \quad (802)$$

where D is the largest dimension of the near-field shape and λ is the free space wavelength. For example, given a near-field shape of dimension $2 \times 2 \mu\text{m}$ and a wavelength of $1 \mu\text{m}$, the far field occurs at the observation distance ($r \geq 8$) μm .

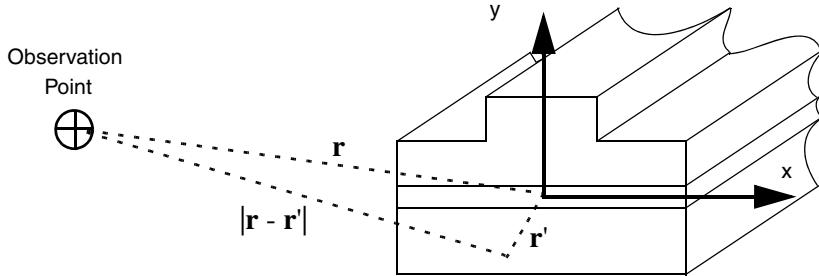


Figure 97 Origin is conveniently placed at center of laser end facet so that $z'=0$; distance between the observation point and a source point on laser end facet is $|r - r'|$

Figure 97 is referred to for the coordinate used in the following derivation of the vectorial far field. The optical electric field at the observation distance r [9] is:

$$E(r) = -\int [\nabla \times \vec{G}(r, r') \bullet M(r')] dr' \quad (803)$$

where \vec{G} is the dyadic Green's operator and M is an equivalent magnetic source representing the near field, $E_{\text{near}}(r')$, of the laser mode:

$$M(r') = -2\hat{z} \times E_{\text{near}}(r') \quad (804)$$

In the far field, it is assumed that the radiation becomes plane waves, and the curl of the dyadic Green's operator can be simplified to:

$$\nabla \times \vec{G}(r, r') = ik\hat{r} \times \vec{G}(r, r') \quad (805)$$

where the unit vector:

$$\hat{r} = \hat{x} \sin \theta \cos \phi + \hat{y} \sin \theta \sin \phi + \hat{z} \cos \theta \quad (806)$$

In addition, the following approximations are assumed for far-field calculations:

$$|r - r'| \approx r \quad \text{for amplitude}$$

$$r - r' \approx r - \hat{r} \bullet r' \quad \text{for phase}$$

With these assumptions, formulas for the vectorial and scalar far-field calculations can be derived.

29: Optical Mode Solver for Lasers

Far Field

In the vectorial case, simplifying the dyadic Green's operator with the scalar Green function and setting $z' = 0$, the vectorial far field can be derived as:

$$E(r, \theta, \phi) = -\frac{ik e^{ikr}}{4\pi r} \left\{ \hat{x} 2 \cos \theta \iint E_{\text{near}(x)}(x', y') e^{-ik\hat{r} \bullet \vec{r}'} dx' dy' + \hat{y} 2 \cos \theta \iint E_{\text{near}(y)}(x', y') e^{-ik\hat{r} \bullet \vec{r}'} dx' dy' - \hat{z} 2 \sin \theta \iint [E_{\text{near}(x)}(x', y') \cos \phi + E_{\text{near}(y)}(x', y') \sin \phi] e^{-ik\hat{r} \bullet \vec{r}'} dx' dy' \right\} \quad (807)$$

where $E_{\text{near}(x)}$ and $E_{\text{near}(y)}$ are the x- and y-components of the near field, respectively.

Using the transformations:

$$\sin \Theta_x = \sin \theta \cdot \cos \phi \quad (808)$$

$$\sin \Theta_y = \sin \theta \cdot \sin \phi \quad (809)$$

you can derive from [Eq. 807](#) the commonly used expression for the constant radius, relative far-field intensity:

$$I_{\text{far}}(\Theta_x, \Theta_y) = (1 - (\sin^2 \Theta_x + \sin^2 \Theta_y)) \left| \iint \Phi(x, y) e^{ik_0(x \sin \Theta_x + y \sin \Theta_y)} dx dy \right|^2 \quad (810)$$

In [Eq. 810](#), it is clear that the scalar near field, Φ , can represent either $E_{\text{near}(x)}$ (TE mode) or $E_{\text{near}(y)}$ (TM mode) in [Eq. 807](#).

NOTE If the square root of the optical intensity is used to compute the far field, the phase information will be lost and the far field will be incorrect. Therefore, it is important to ensure that full scalar or vectorial near-fields are used in the far-field calculations.

Far-Field Observation Angle

The mapped far-field observation angles, (Θ_x, Θ_y) , refer to the angles measured from the z-axis along the x-axis and y-axis, respectively. The laser end facet (location of the near field) is assumed to be at the origin, facing the direction of positive z (see [Figure 98 on page 789](#)).

The far-field observation distance is fixed at a constant radius from the origin where the device is, that is, the observation space is a hemisphere. As a result, constraints must be imposed on (Θ_x, Θ_y) because the aim is to map a rectangular (Θ_x, Θ_y) $[-\pi/2, \pi/2]$ space onto a hemisphere. The best way to visualize this constraint is to think of placing a square piece of fishnet over a hemisphere. The corner regions of the net will exceed the boundary of the

hemisphere and, hence, do not contribute to the description of the hemisphere. These are the invalid zones of the observation space.

NOTE In the far-field calculation, the origin is aligned with the peak intensity of the fundamental mode of the edge-emitting laser, and the observation angles are defined with respect to this origin.

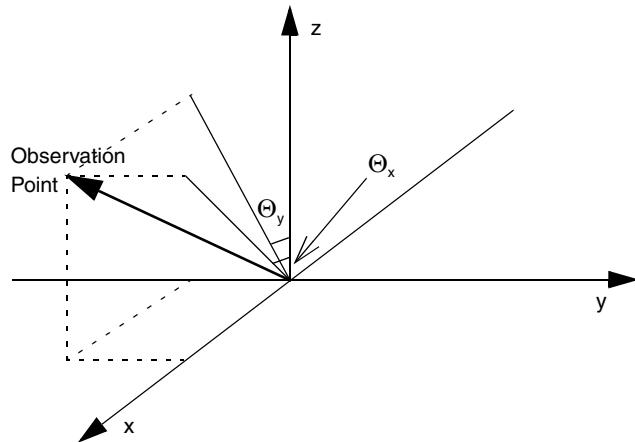


Figure 98 Defining the observation angles for far field

Syntax for Far Field

The optical far field is computed and plotted if the keyword `OptFarField` is specified in the `File` section of the command file:

```
File {...  
    # ----- Activate farfield computation and save -----  
    OptFarField = "far"  
}  
FarFieldPlot{range=(40,60) intervals=40 Scalar1D Scalar2D Vector2D}  
...  
Solve {...  
  
    # ----- Farfield plot parameters must be specified inside quasistationary -  
----  
    quasistationary {...  
  
        PlotFarField{range=(0,1) intervals=3}  
  
        Goal {name="p_Contact" voltage=1.8}  
        {...}  
    }  
}
```

29: Optical Mode Solver for Lasers

Far Field

See [Simulating Single-Grid Edge-emitting Laser on page 658](#) for a clearer picture of the placement of the far-field syntax inside the command file. The far field syntax works in the same way as the GainPlot (see [Plotting Gain and Power Spectra on page 855](#)) and Plot options in the Quasistationary statement.

The more notable features of the syntax are:

- The base file name "far" of the far-field files is specified by OptFarField in the File statement. The keyword OptFarField also activates the far-field plot.
- The far field can only be computed and plotted within the Quasistationary statement. The keyword PlotFarField and the FarFieldPlot statement control the number and type of far field plots to produce.
- The argument range=(0, 1) in the PlotFarField keyword is mapped to the initial and final bias conditions. In this example, the initial and final (goal) p_Contact voltages are 0 V and 1.8 V, respectively. The number of intervals=3, which gives a total of four (= 3+1) far-field plots at 0 V, 0.6 V, 1.2 V, and 1.8 V. In general, specifying intervals=n will produce (n+1) plots.
- The argument range=(40, 60) in the FarFieldPlot statement is the range for the observation angles. In this example, $\Theta_x = [-20^\circ, 20^\circ]$ and $\Theta_y = [-30^\circ, 30^\circ]$ were chosen. The number of discretized points in each range of the observation angles is given by intervals=40.
- You can select any one or a combination of the three types of far-field plot: Scalar1D, Scalar2D, and Vector2D. These keywords correspond to the scalar one-dimensional far field, scalar two-dimensional far field, and vectorial two-dimensional far field, respectively. The different types of far-field plot will generate different types of output files.

[Table 127 on page 1091](#), [Table 139 on page 1105](#), and [Table 228 on page 1181](#) give the activating syntaxes for the far field.

Far-Field Output Files

Activating different types of far-field plot produces different output files. Referring to the example syntax listed in [Syntax for Far Field on page 789](#), where the far-field base name has been specified by OptFarField="far", output examples where different far-field types are selected are shown.

Scalar1D Far Field

The scalar 1D far field is activated by the keyword Scalar1D. Two lines per mode are produced for the 1D far-field plot. One is the far-field intensity measured along Θ_x with $\Theta_y = 0$; the other, along Θ_y with $\Theta_x = 0$. These far field results are output in the files

`far_ff_000000_des.plt`, `far_ff_000001_des.plt`, and so on at the requested bias intervals specified in the argument of `PlotFarField`.

These files can be viewed in Inspect and an example of the plot is shown in [Figure 99](#).

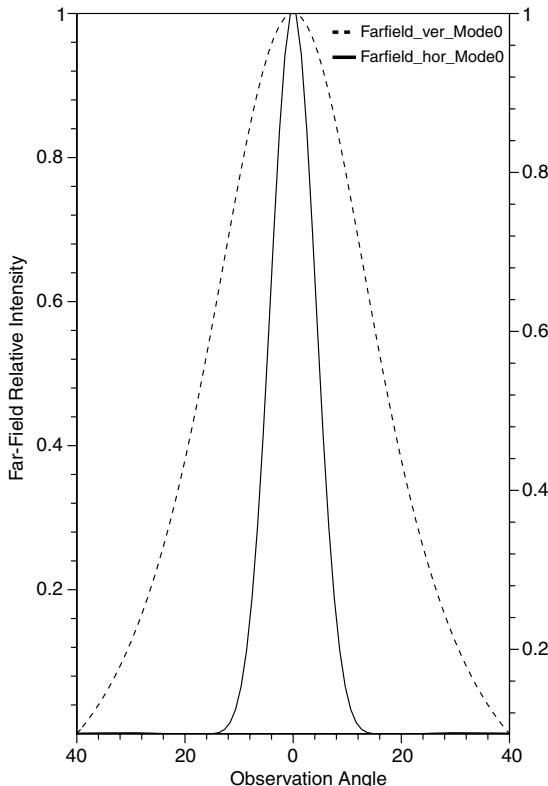


Figure 99 Scalar 1D far-field patterns showing vertical (*dashed line*) and horizontal (*solid line*) variations

Scalar2D and Vector2D Far Fields

The 2D scalar and vectorial far fields are activated by the keywords `Scalar2D` and `Vector2D`, respectively. An observation angle grid, `far_ff_des.grd`, is created if either keyword is detected. The vectorial far field can only be activated if the vectorial optical solver has been chosen or a vectorial mode is input from a file.

The data file for the scalar 2D far field contains the normalized far-field pattern of each mode, and the file names are `far_ff_des_000000_Scalar.dat`, and so on. Each file contains as many far-field patterns as the number of modes in the simulation. If a vectorial optical solver has been chosen, Sentaurus Device automatically selects the strongest field component (either E_x or E_y) for its scalar far-field computation.

29: Optical Mode Solver for Lasers

Far Field

The data file for the vectorial 2D far field contains the normalized far-field pattern for the x-, y-, and z-components and the total far field for each mode. These data files are named as `far_ff_des_000000_Vector.dat`, and so on. The 2D scalar and vectorial far fields can be viewed in Tecplot SV like any other `.grd` and `.dat` files. Examples of these far-field patterns are shown in [Figure 100](#) and [Figure 101](#).

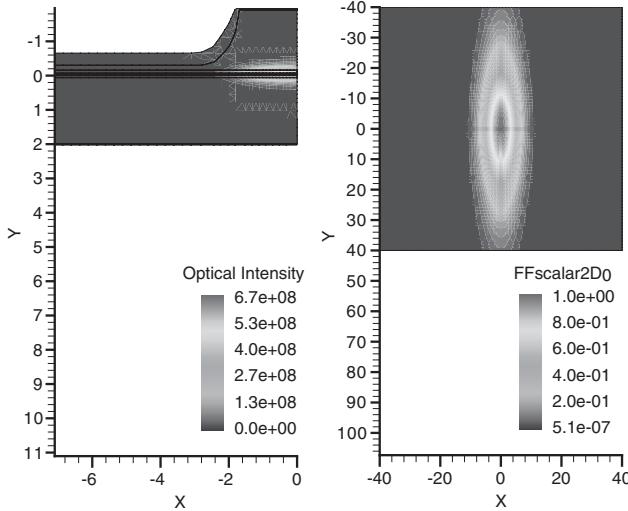


Figure 100 Near field (left) and scalar 2D far field (right) of edge-emitting laser

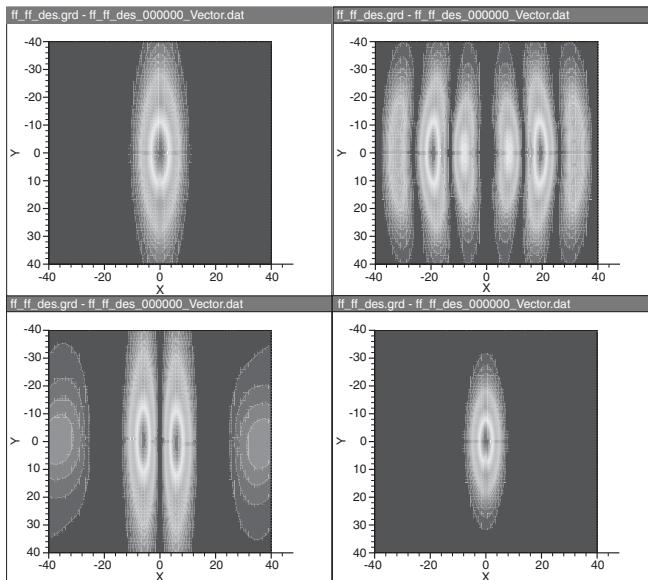


Figure 101 The x (upper left), y (upper right), z (lower left) components, and total absolute value (lower right) of vectorial 2D far fields of edge-emitting laser; x-component is a few orders of magnitude stronger than y- and z-components, so the absolute of the vectorial far field resembles that of x-component

Far Field from Loaded Optical Field File

Far fields can be computed from the optical mode loaded into Sentaurus Device (see [Saving and Loading Optical Modes on page 783](#)). To compute the far field correctly, the full scalar or vectorial optical near field with the complete phase information must be loaded. The optical intensity does not contain phase information and will give an incorrect far-field pattern.

VCSEL Near Field and Far Field

While the near field of an edge emitter is apparent, the location of the near field of a VCSEL is not automatically detected in Sentaurus Device. Different VCSELs have different geometric designs and the entire top surface may not be the emitting surface. Therefore, in the case of VCSELs, you must specify the location of the near field that will be used to compute the far field.

The activating syntaxes for the VCSEL near field are in the `File`, `VCSELNearFieldPlot`, and `Solve-quasistationary` sections of the command file:

```
File {...  
    VCSELNearField = "vcselnf"  
}  
...  
Physics {...  
    Laser (...  
        Optics (...  
            FEVectorial(...)  
        )  
        VCSEL ()  
    )  
}  
VCSELNearFieldPlot(Position=0.0 Radius=10.0 NumberOfPoints=100)  
# (<z> <radius> Npoints) [micrometers]  
...  
Solve {...  
    # ----- VCSEL near field parameters must be specified inside quasistationary  
-----  
    quasistationary (...  
        VCSELNearField { range=(0,1) intervals=3 }  
        Goal({name="p_Contact" voltage=1.8})  
    {...}  
}
```

29: Optical Mode Solver for Lasers

VCSEL Near Field and Far Field

An analysis of the syntax is:

- In the `File` section, the VCSEL near field is activated by the keyword `VCSELNearField`, and the value assigned to it, "vcselnf" in this case, is the base name for the output files of the near field.
- In the `VCSELNearFieldPlot` statement, the location and discretization mesh of the near field is defined by the keywords `Position=<z>`, `Radius=<radius>`, and `NumberOfPoints=<n>`. In VCSELs where cylindrical geometry has been assumed, `Radius` and `Position` (in micrometers) define the location of the near field, that is, the near field is taken from $(0, <z>)$ to $(<radius>, <z>)$. `n` defines the number of points on each side of the square mesh where the near field will be plotted.
- In the `Solve-quasistationary` section, the argument `range=(0,1)` in the `VCSELNearField` keyword is mapped to the initial and final bias conditions. In this example, the initial and final (goal) `p>Contact` voltages are 0 V and 1.8 V, respectively. The number of `intervals=3`, which gives a total of four ($= 3+1$) near field plots at 0 V, 0.6 V, 1.2 V, and 1.8 V. In general, specifying `intervals=n` will produce $(n+1)$ plots.
- The square mesh grid file for the near field will be named `vcselnf.grd`. At each requested bias output, a file named `vcselnf_xxx_des.dat` will be created. For each mode, it contains the datasets `Abs(modeX_real_OpticalField)`, `Abs(modeX_imag_OpticalField)`, and `modeX_OpticalIntensity`. The near fields can be viewed in Tecplot SV.
- Cylindrical symmetry has been assumed so that the optical fields decompose into cylindrical harmonics, $e^{im\phi}$, where m is the cylindrical harmonic order defined by either the keyword `AzimuthalExpansion` for FEVectorial VCSEL simulation or the keyword `mModeIndex` for EffectiveIndex VCSEL simulation. The VCSEL near field can be divided into the even part and the odd part. Therefore, the field components are multiplied by either $\cos(m\phi)$ or $\sin(m\phi)$. Sentaurus Device plots the even part of the VCSEL near field.

When the VCSEL near field is computed, the activation of the far-field calculations is similar to that of other laser simulations.

NOTE In the case of VCSELs, the far field is not computed unless the near field is specified.

Optics Stand-alone Option

In many cases, the initial phase of design of a laser diode involves the optimization of the geometry of the laser structure to achieve specific optical mode properties, for example, mode shape. Sentaurus Device provides an option to run the optical solver without invoking the entire laser simulation.

An example of a stand-alone optical simulation of an edge-emitting laser, waveguiding structure is:

```
# ----- Optics Stand-alone command file -----

# ----- Specify only a dummy electrode -----
Electrode {
    { Name="dummy"    voltage=0.0 }
}

# ----- Tell Sentaurus Device where to read/save the parameters/results -----
File {
    Grid = "optmesh_mdr.grd"
    Doping = "optmesh_mdr.dat"
    Parameter = "des_las.par"
    Plot = "opt_plot"
    Output = "log"
    # ----- Compute and save the farfield -----
    OptFarField = "farfield"
    # ----- Save the optical field -----
    SaveOptField = "optmode"
}

Plot {
    RefractiveIndex
}

# ----- Specify the material region properties, as usual -----
Physics (region="pbulk") { MoleFraction(xfraction=0.28) }
Physics (region="nbulk") { MoleFraction(xfraction=0.28) }
Physics (region="psch") { MoleFraction(xfraction=0.09) }
Physics (region="nsch") { MoleFraction(xfraction=0.09) }
Physics (region="barr") { MoleFraction(xfraction=0.09) }
# ----- Quantum Wells -----
Physics (region="qw1") {
    MoleFraction(xfraction = 0.8 Grading=0.00)
    Active      # keyword to specify active region
}
Physics (region="qw2") {
    MoleFraction(xfraction = 0.8 Grading=0.00)
```

29: Optical Mode Solver for Lasers

Optics Stand-alone Option

```
        Active
    }

# ----- Major difference from the laser command file -----
Physics {
    Optics (
        FEVectorial (
            EquationType = Waveguide      # or Cavity
            Symmetry = Nonsymmetric     # or Symmetric or Periodic
            LasingWaveLength = 656       # [nm]
            TargetEffectiveIndex = 3.45
            Boundary = "Type2"
            ModeNumber = 1
        )
    )
    HeteroInterface
}

FarFieldPlot{ Range=(30,40) Intervals=60 }

Solve { Optics }
```

Compare this code to [Simulating Single-Grid Edge-emitting Laser on page 658](#). The most significant difference in the optics stand-alone simulation is that the entire Laser section has been removed. Other notable changes are:

- In the `Electrode` statement, a dummy electrode must be specified in order to conform to the input format of Sentaurus Device. It has no other purpose in the optical simulation.
- In the `File` section, you can choose to plot the far field with the keyword `OptFarField`. The far-field parameters are set in the `FarFieldPlot` section.
- In the `File` section, you must specify the base name for the optical-field files with the keyword `SaveOptField`. These files can be read into the laser simulation (see [Saving and Loading Optical Modes on page 783](#)).
- The optical-active region is specified in the material region `Physics` statement by the keyword `Active`. This is important so that the optical confinement factor can be computed within Sentaurus Device.
- In the `Optics` section, the optical mode solver type (`FEScalar`, `FEVectorial`, `TMM1D`, or `EffectiveIndex`) is specified. The arguments for these mode solver types are discussed in:
 - [Syntax of FE Scalar and FE Vectorial Optical Solvers on page 763](#)
 - [Transfer Matrix Method for VCSELs on page 775](#)
 - [Effective Index Method for VCSELs on page 777](#)

- Optics stand-alone simulation does not support multimode simulation, that is, `ModeNumber=1`.
- In the `Solve` section, the stand-alone calculation of the optical mode solver is activated by the sole keyword `Optics`.

Automatic Optical Mode Searching

The cavity and waveguide mode solvers in Sentaurus Device require you to choose an initial guess so that a sparse matrix solver can be used to find the correct optical mode. It is not always easy to estimate a good initial guess. To circumvent this problem, a model has been developed to help identify the probability of physical modes through a mode density calculation. The concept is simple: A Hertzian dipole is placed in the cavity or waveguide, and the deterministic equations:

$$\nabla \times (\nabla \times \bar{E}) - \frac{\omega^2}{c^2} \epsilon_r \bar{E} = -i\omega\mu \cdot \vec{j}_{\text{src}}(r, \omega) \quad (811)$$

are solved for the cylindrically symmetric cavity resonance problem and:

$$\nabla \times (\nabla \times \bar{E}) - \frac{\omega^2}{c^2} \epsilon_r(x, y) \cdot \bar{E}(x, y) + \gamma^2 \cdot \bar{E}(x, y) = -i\omega\mu \cdot \vec{j}_{\text{src}}(r, \omega) \quad (812)$$

are solved for the waveguide problem.

The source dipole $j_{\text{src}}(r, \omega)$ is placed strategically in the cavity (see [Figure 102](#)) or waveguide (see [Figure 103 on page 798](#)).

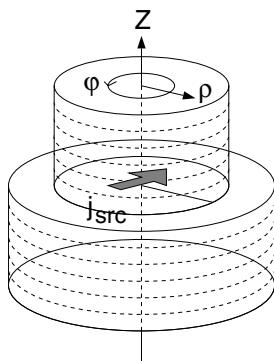


Figure 102 Dipole source in a cylindrically symmetric cavity problem can be radially or azimuthally defined

29: Optical Mode Solver for Lasers

Automatic Optical Mode Searching

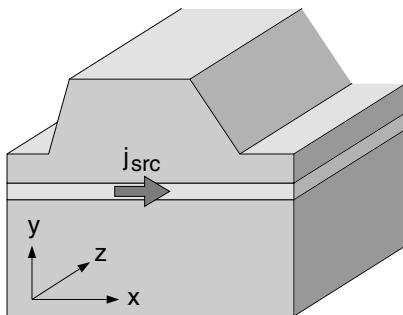


Figure 103 Dipole source in a waveguide problem can be orientated in x-direction or y-direction to excite TE and TM modes

The source dipole will excite a plethora of modes containing guided, resonating, and leaky modes. By sweeping through the wavelengths for the cavity problem or the effective indices for the waveguide problem, the change of energy in the cavity or waveguide at each frequency or effective index can be computed. This energy change is most sensitive near a resonating or guided mode and, therefore, provides the key to identifying the physical resonating or guided modes. The total energy contained in the system is given by:

$$\int \int_{\text{surf}} (\bar{S} \cdot d\bar{A}) = -\frac{1}{2} \iiint_V \{ \sigma \bar{E} \cdot \bar{E}^* + i\omega(\mu \bar{H} \cdot \bar{H}^* + \epsilon \bar{E} \cdot \bar{E}^*) \} dV \quad (813)$$

The real part gives the physically dissipated energy of the system and the imaginary part gives the energy stored in the system. In any system containing resonances, it is well known that the energy dissipation at the resonance point reduces sharply to a minimum. Therefore, by tracking the changes in dissipated energy, it is possible to estimate the resonant wavelength or effective index of the required mode.

There are remote possibilities that the dipole source is placed at the null of the required mode. In this case, the mode will not be excited and will not show up in the plot of the change in dissipation energy. You can place the dipole source at alternative positions to ensure that the required mode is excited.

Syntax for Automatic Mode Searching

There are some differences in the syntax for the automatic mode search for the cavity and waveguide problems. [Table 186 on page 1152](#) gives a summary of the keywords for `DeterministicProblem()`.

Searching for Cavity Resonances

Cylindrical symmetry is assumed for the cavity problem. This is applicable to the case of cylindrically symmetric VCSEL structures. The syntax for setting up the deterministic search for cavity resonances is:

```
File {
    ...
    SaveOptSpectrum = "spectrum"
}

...
Physics {
    Laser (
        Optics (
            DeterministicProblem (
                EquationType = Cavity
                Coordinates = Cylindrical
                AzimuthalExpansion = 1
                SourcePosition (0.25 0.5)      # [um]
                SourcePolarization = rhoDirection
                Sweep (740.0 760.0 80)       # [nm]
            )
        )
        VCSEL()
    )
    ...
}
.....
Solve { Optics }
```

The SourcePolarization of the dipole can be chosen to be in different directions. The range of wavelengths to search is given in units of nanometers. An example of the output plot is shown in [Figure 104 on page 800](#). As expected, the resonances occur at locations with sharp energy changes.

29: Optical Mode Solver for Lasers

Automatic Optical Mode Searching

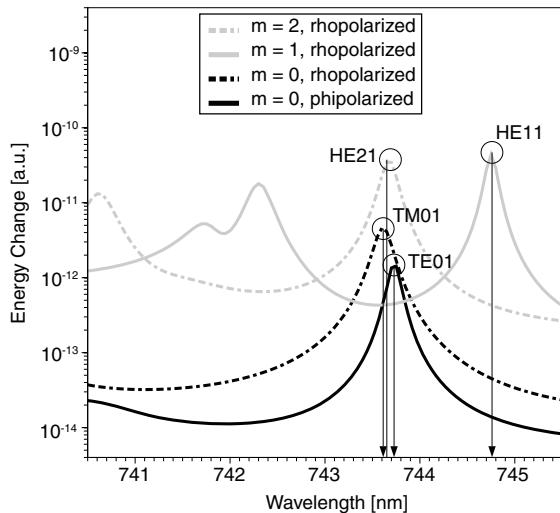


Figure 104 Energy changes plotted as a function of wavelength for the cavity problem; resonances of the different modes are clearly identified

Searching for Waveguide Modes

Searching for waveguide modes requires sweeping through the effective indices, and the syntax for this is:

```
File {
    SaveOptSpectrum = "spectrum"
}
...
Physics {
    Laser (
        Optics (
            DeterministicProblem (
                EquationType = Waveguide
                Symmetry = Symmetric
                Boundary = Type1
                Coordinates = Cartesian
                LasingWavelength = 530# [nm]
                SourcePosition (1.0 1.533)
                SourcePolarization = xDirection
                Sweep (3.391 3.401 100)
            )
        )
    )
}
...
Solve { Optics }
```

An example of the output is shown in Figure 105.

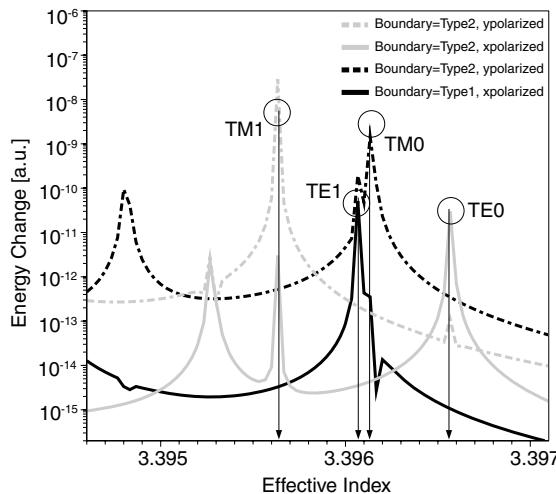


Figure 105 Energy changes plotted as a function of the effective index; guided modes of the waveguide are clearly identified

References

- [1] J. Jin, *The Finite Element Method in Electromagnetics*, New York: John Wiley & Sons, 2nd ed., 2002.
- [2] G. L. G. Sleijpen *et al.*, “Jacobi-Davidson Type Methods for Generalized Eigenproblems and Polynomial Eigenproblems,” *BIT*, vol. 36, no. 3, pp. 595–633, 1996.
- [3] D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst, “Jacobi–Davidson Style QR and QZ Algorithms for the Reduction of Matrix Pencils,” *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 94–125, 1998.
- [4] M. Streiff *et al.*, “A Comprehensive VCSEL Device Simulator,” *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 9, no. 3, pp. 879–891, 2003.
- [5] W. C. Chew and W. H. Weedon, “A 3D Perfectly Matched Medium from Modified Maxwell’s Equations with Stretched Coordinates,” *Microwave and Optical Technology Letters*, vol. 7, no. 13, pp. 599–604, 1994.
- [6] J.-P. Berenger, “A Perfectly Matched Layer for the Absorption of Electromagnetic Waves,” *Journal of Computational Physics*, vol. 114, no. 2, pp. 185–200, 1994.
- [7] F. L. Teixeira and W. C. Chew, “Systematic Derivation of Anisotropic PML Absorbing Media in Cylindrical and Spherical Coordinates,” *IEEE Microwave and Guided Wave Letters*, vol. 7, no. 11, pp. 371–373, 1997.

29: Optical Mode Solver for Lasers

References

- [8] F. L. Teixeira and W. C. Chew, “A General Approach to Extend Berenger’s Absorbing Boundary Condition to Anisotropic and Dispersive Media,” *IEEE Transactions on Antennas and Propagation*, vol. 46, no. 9, pp. 1386–1387, 1998.
- [9] S. L. Chuang, *Physics Of Optoelectronic Devices*, New York: John Wiley & Sons, 1995.

This chapter presents the physics of carrier scattering (capture) into quantum wells, meshing restrictions for the quantum wells, and methods of gain calculations for the quantum wells and bulk regions.

These gain methods include the simple rectangular well model, the advanced $k \cdot p$ method (both zinc-blende and wurtzite crystal structures), and automatic loading of precomputed gain tables. Various gain-broadening mechanisms, strain effects, and nonlinear gain saturation effects are discussed. An advanced piezoelectric field screening model for GaN-type quantum wells is also presented.

Overview

The drift-diffusion transport phenomena contained in the continuity and hydrodynamic equations are not suitable for modeling the transport across the quantum well (QW) because the feature size of the quantum well is much smaller than the inelastic mean free path of the carrier.

In this section, the focus is on three aspects of modeling the quantum well:

- Carrier capture into the quantum well
- Radiative recombination processes important in a quantum well
- Gain calculations

The QW carrier capture process is treated with a ballistic approach. A few types of recombination processes are important in the quantum well:

- Auger and Shockley–Read–Hall (SRH) recombinations deplete the QW carriers, and they form the dark current.
- Radiative recombination contains the stimulated and spontaneous recombination processes, which are important processes in lasers and LEDs.

These recombinations must be added to the carrier continuity equations to ensure the conservation of particles.

The gain calculation is based on Fermi's golden rule and describes quantitatively the radiative emissions in the form of the stimulated and spontaneous emission coefficients. These coefficients contain the optical matrix element $|M_{ij}|^2$, which describes the probability of the radiative recombination processes. In the quantum well, computing the optical matrix element

requires knowledge of the QW subbands and QW wavefunctions. The screening of piezoelectric fields in GaN-based quantum wells by the carriers contained in the wells can also be taken into account (see [Screening Piezoelectric Fields in GaN-type Quantum Wells on page 839](#)).

Sentaurus Device offers three options for computing the gain spectrum:

- A simple finite well model with analytic solutions. In addition, strain effects and polarization dependence of the optical matrix element are handled separately.
- The $k \cdot p$ method, which includes strain and manybody effects with its 4×4 , 6×6 , or 8×8 Hamiltonian matrix.
- User-specified gain routines in C++ language can be coupled self-consistently with Sentaurus Device using the physical model interface (PMI).

Carrier Capture in Quantum Wells

A ballistic transport approach is used to handle the carrier capture or escape into or out of a quantum well. It follows the treatment of Grupen and Hess [1]. This approach is illustrated in [Figure 106](#).

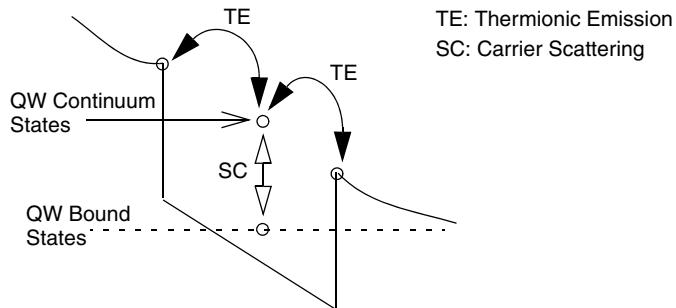


Figure 106 Discretization of quantum well to handle the physics of quantum well transport

The transport perpendicular to the QW plane is treated as follows. The QW is treated as a point source of recombination, which contains separate continuum and bound states. In this case, the continuum and bound states are assumed to have different quasi-Fermi levels that lead to separate continuity equations for the continuum and bound states. Transport of carriers from the regions outside the QW to the QW continuum states is by thermionic emission. The transition from the QW continuum to the bound states is computed by a scattering rate that includes carrier–carrier scattering. The bound states are solved from the Schrödinger equation.

Within the quantum well in the direction parallel to the QW plane, drift-diffusion transport is assumed to be valid.

Special Meshing Requirements for Quantum Wells

As a result of the transport in [Figure 106 on page 804](#), the spatial discretization of the mesh at the quantum well must be treated specially. The discretization is based on a ‘three-point’ model in the direction perpendicular to the QW plane. Only one vertex is placed in the quantum well and one vertex is at each quantum well–bulk interface (see [Figure 107](#)). In the quantum well, only rectangular elements are allowed. This must be ensured when building the mesh.

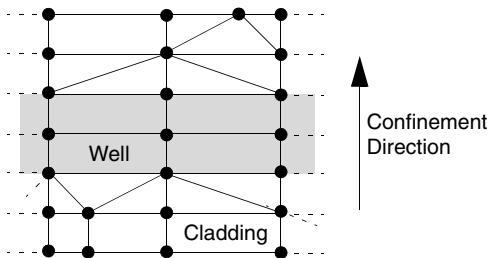


Figure 107 Discretization of quantum well

NOTE The discretization constraint for the quantum well can be chosen by creating a refinement region at the quantum well in the mesh generator.

Thermionic Emission

Thermionic emission is used at the quantum well interface to handle the large momentum changes caused by the band-edge discontinuity. If thermionic emission is used only, without the QW scattering model, all the carriers are assumed to be totally captured in the quantum well.

The default setting uses thermionic emission at the quantum well interfaces. This is activated by default if the keyword `QWTransport` is used in the `Physics-Laser` section of the command file:

```
Physics {...  
  Laser (...  
    Optics (...)  
    # ----- Specify QW model and physics -----  
    QWTransport  
    QWExtension = AutoDetect # QW widths auto detection  
  )  
}
```

30: Modeling Quantum Wells

Carrier Capture in Quantum Wells

The QW region must be identified by the keyword `Active` (see [Simulating Single-Grid Edge-emitting Laser on page 658](#)) and must be discretized according to the constraints of the special ‘three-point’ QW model. The keyword `QWTransport` informs Sentaurus Device to use the ‘three-point’ QW model to treat the `Active` quantum well region.

It is possible to use the thermionic emission QW model without the QW scattering model for isothermal quasistationary simulations, in which case, all carriers are assumed to be totally captured in the quantum well. However, excluding the QW scattering model in nonisothermal simulations (that is, when the hydrodynamic or thermodynamic temperature equations are included) will lead to convergence problems.

Originally, the laser simulation models in Sentaurus Device were developed for devices in which the lateral leakage currents across quantum wells, that is, nonperpendicular current flux of free carriers across quantum wells had not been taken into account. However, for devices that require the inclusion of such currents in the simulation, the keyword `LateralLeakage` can be used in the `Physics-Laser` section to take into account these current components. The total leakage currents for electrons and holes, that is, the difference between the total currents and the net carrier capture rates (see [Quantum-Well Scattering Model](#)) on active vertices can be plotted by including the keywords `QW_eLeakageCurrent` and `QW_hLeakageCurrent` in the `Plot` section of the command file.

Quantum-Well Scattering Model

A physically intuitive model is used to handle the physics of carrier scattering at the quantum well. The carrier populations are separated into bound and continuum states, and separate continuity equations are applied to both populations. The QW scattering model accounts for the net capture rate, that is, not all of the carriers will be scattered into the bound states of the quantum well.

The electron capture rate from the continuum (subscript 3) to the bound (subscript 2) states is:

$$R = \int_{E_C}^{\infty} dE_3 \int_{E_{\text{well}}}^{\infty} dE_2 \cdot g_3(E_3)g_2(E_2)S(E_2, E_3)f_3(E_3)(1-f_2(E_2)) \quad (814)$$

where $g(E)$ is the density-of-states, $S(E_2, E_3)$ is the scattering probability, and $f(E)$ is the Fermi-Dirac distribution.

The reverse process gives the electron emission rate from the bound to continuum states:

$$M = \int_{E_C}^{\infty} dE_3 \int_{E_{\text{well}}}^{\infty} dE_2 \cdot g_3(E_3)g_2(E_2)S(E_3, E_2)f_2(E_2)(1-f_3(E_3)) \quad (815)$$

Therefore, the net capture rate is [1]:

$$C = R - M = (1 - e^{\eta_2 - \eta_3}) \left(1 - \frac{n_2}{N_2} \right) \frac{n_3}{\tau} \quad (816)$$

where:

$$N_2 = \int_{E_W}^{E_c} dE_2 g_2(E_2) \quad (817)$$

$\eta_2 = (-q\Phi_2 - E_C)/k_B T$ and $\eta_3 = (-q\Phi_3 - E_C)/k_B T$ contain the quasi-Fermi level information and τ is the capture time. The capture time is representative of the carrier–carrier and carrier-optical phonon scattering processes when the carriers are scattered into the quantum well.

For very deep quantum wells, the keyword `QWDeep` activates a modified capture model in which the capture rates are give as:

$$C = R - M = (1 - e^{\eta_2 - \eta_3}) \frac{n_3}{\tau} \quad (818)$$

For shallow quantum wells, the energy transfer during scattering can only occur in a limited range. In the limit of elastic scattering, the net capture rate for shallow quantum wells is:

$$C = \left(\frac{F_{3/2}(\eta_3)}{F_{1/2}(\eta_3)} - \frac{F_{3/2}(\eta_2)}{F_{1/2}(\eta_2)} \right) \frac{\eta_3}{\tau} \quad (819)$$

where F_m is the Fermi integral of the order of m . The shallow quantum well model is especially suitable for InGaAsP-type quantum wells that have smaller conduction and valence band offsets compared to InGaAs and GaAs wells. It is activated by the keyword `QWSHallow`.

This capture rate is added to the continuity equations as a recombination term. The treatment for holes is similar. It is apparent that separate capture times must be given for the electrons and holes accordingly, and these can be specified by the keywords `QWeScatTime` and `QWhScatTime` in the `Physics-Laser` section of the command file.

The activating syntax for the QW scattering model is located in the `Physics-Laser` and `Solve` sections of the command file:

```
Physics {...  
  Laser (...  
    Optics (...)  
    # ---- Use 3-point QW model ----  
    QWTransport  
    QWEextension = AutoDetect  
    # ---- Activate QW scattering model ----
```

30: Modeling Quantum Wells

Carrier Capture in Quantum Wells

```
QWScatModel
QWeScatTime = 8e-13      # [s]
QWhScatTime = 4e-13      # [s]
eQWMobility = 1450        # [cm^2/Vs]
hQWMobility = 370         # [cm^2/Vs]
# QWShallow               # for shallow QWs only
)
}
...
Solve {
    Coupled {Poisson}
    Coupled {Poisson Electron Hole}
    Coupled {Poisson Electron Hole QWeScatter QWhScatter}
    Coupled {Poisson Electron Hole QWeScatter QWhScatter PhotonRate}
...
}
```

The net capture rates for electrons and holes can be plotted by including the keywords `QW_eNetCapture` and `QW_hNetCapture` in the `Plot` section of the command file.

Some comments about the syntax are:

- Switching on the QW scattering model with `QWScatModel` requires that the keyword `QWTransport` is included to set up the ‘three-point’ QW model.
- The keyword `QWEextension=AutoDetect` activates the automatic extraction of quantum-well thickness according to the keyword `Active` in the material region `Physics` section. This is the default behavior. If all quantum wells have the same thickness, the extracted values can be overridden by specifying `QWEextension=<float>` (in nanometers).
- `QWShallow` should only be activated for shallow quantum wells.
- In the `Solve` section, equation systems are added consecutively. Each `Coupled` solution provides the initial guesses for the next `Coupled` problem containing more equation systems. In this case, `Coupled {Poisson Electron Hole}` solves the quantum well transport with thermionic emission only. The solution provides a good initial guess for the next `Coupled` problem when the QW scattering models are added.
- The QW scattering model creates the continuity equations for the bound carriers, so you must input the QW bound carrier mobilities with the keywords `eQWMobility` and `hQWMobility`.

Radiative Recombination and Gain Coefficients

After the carriers are captured in the active region, they experience either dark recombination processes (such as Auger and SRH) or radiative recombination processes (such as stimulated and spontaneous emissions), or escape from the active region. This section describes how stimulated and spontaneous emissions are computed in Sentaurus Device.

Stimulated and Spontaneous Emission Coefficients

In the active region of the laser, radiative recombination is treated locally at each active vertex. The stimulated and spontaneous emissions are computed using Fermi's golden rule.

At each active vertex of the quantum wells, the local stimulated emission coefficient is:

$$r^{\text{st}}(\hbar\omega) = \sum_{i,j} \int dE C_o k_{\text{st}} |M_{i,j}|^2 D(E) \times (f_i^C(E) + f_j^V(E) - 1) L(E) \quad (820)$$

and the local spontaneous emission coefficient is:

$$r^{\text{sp}}(\hbar\omega) = \sum_{i,j} \int dE C_o k_{\text{sp}} |M_{i,j}|^2 D(E) f_i^C(E) f_j^V(E) L(E) \quad (821)$$

where:

$$C_0 = \frac{\pi e^2}{n_g c \epsilon_0 m_0^2 \omega} \quad (822)$$

$$|M_{i,j}|^2 = P_{ij} |O_{i,j}|^2 \left(\frac{m_0}{m_e} - 1 \right) \frac{m_0 E_g (E_g + \Delta)}{12 \left(E_g + \frac{2}{3} \Delta \right)} \quad (823)$$

$$O_{i,j} = \int_{-\infty}^{\infty} dx \zeta_i(x) \zeta_j^*(x) \quad (824)$$

$$f_{(i, \Phi_n, E)}^C = \left(1 + \exp \left(\frac{E_C + E_i + q\Phi_n + \frac{m_r}{m_e} E}{k_B T} \right) \right)^{-1} \quad (825)$$

30: Modeling Quantum Wells

Radiative Recombination and Gain Coefficients

$$f_{(j, \Phi_p, E)}^V = \left(1 + \exp \left(\frac{E_V - E_j + q\Phi_p - \frac{m_r}{m_h}E}{k_B T} \right) \right)^{-1} \quad (826)$$

$$D_{(E)}^r = \frac{m_r}{\pi \hbar^2 L_x} \quad (827)$$

$$m_r = \left(\frac{1}{m_e} + \frac{1}{m_h} \right)^{-1} \quad (828)$$

$L(E)$ is the gain-broadening function. The electron, light-hole, and heavy-hole subbands are denoted by the indices i and j . $f_{i(E)}^C$ and $f_{j(E)}^V$ are the local Fermi–Dirac distributions for the conduction and valence bands, $D_{(E)}^r$ is the reduced density-of-states, $|O_{i,j}|^2$ is the overlap integral of the quantum mechanical wavefunctions, and P_{ij} is the polarization-dependent factor of the momentum (optical) matrix element $|M_{i,j}|^2$. The spin-orbit split-off energy is Δ and E_g is the bandgap energy. The anisotropic polarization-dependent factor P_{ij} in the optical matrix element is discussed in [Polarization-dependent Optical Matrix Element on page 822](#). These emission coefficients determine the rate of production of photons when given the number of available quantum well carriers at the active vertex.

k_{st} and k_{sp} are scaling factors for the optical matrix element $|M_{i,j}|^2$ of the stimulated and spontaneous emissions, respectively. They have been introduced to allow you to tune the stimulated and spontaneous gain curves. Consequently, these parameters can change the threshold current.

The activating keywords are `StimScaling` and `SponScaling` in the `Physics-Laser` section of the command file:

```
Physics {...  
  Laser (...  
    Optics (...)  
    # ---- Scale stimulated & spontaneous gain ----  
    StimScaling = 1.0  # default value is 1.0  
    SponScaling = 1.0  # default value is 1.0  
  )  
}
```

As an alternative to the full spontaneous emission model given by [Eq. 821](#), a simpler model is available in Sentaurus Device that is described by:

$$r^{sp}(\hbar\omega) = C \cdot n \cdot p \cdot \left(\frac{T}{T_{par}} \right)^\alpha \quad (829)$$

where C is the coefficient of radiative recombination, n is the electron density, p is the hole density, T is the temperature, T_{par} is the reference temperature, and α is a temperature exponent. The model parameters C , T_{par} , and α must be entered in the RadiativeRecombination section of the parameter file as `C` (with unit [$\text{cm}^3 \text{s}^{-1}$]), `Tpar` (with unit [K]), and `alpha` (with unit [1]), respectively.

The differential gains for electrons and holes are given by the derivatives of the coefficient of stimulated emission $r^{\text{st}}(\hbar\omega)$ (see Eq. 820) with regard to the respective carrier density. They can be plotted by including the keywords `eDifferentialGain` and `hDifferentialGain` in the `Plot` section of the command file.

Active Bulk Material Gain

The stimulated and spontaneous emission coefficients discussed are derived for the quantum well. However, these coefficients can apply to bulk materials with slight modifications. In bulk active materials, it is assumed that the optical matrix element is isotropic. The sum over the subbands is reduced to one electron, and one heavy-hole and one light-hole level, because there is no quantum-mechanical confinement in bulk material. In addition, the subband energies are set to $E_i = 0$, and the following coefficients are modified:

$$O_{i,j} = 1 \quad (830)$$

$$D^r(E) = \frac{1}{2\pi^2} \left(\frac{2m_r}{\hbar^2} \right)^{3/2} E^{1/2} \quad (831)$$

$$P_{i,j} = 1 \quad (832)$$

All other expressions remain the same.

Stimulated Recombination Rate

The radiative emissions contribute to the production of photons but they also deplete the carrier population in the active region. At each active vertex, the stimulated recombination rate of the carriers must be equal to the sum of the photon production rate of every lasing mode so that conservation of particles is ensured. The stimulated recombination rate for each active vertex is:

$$R^{\text{st}}(x, y) = \sum_i r^{\text{st}}(\hbar\omega_i) S_i |\Psi_i(x, y)|^2 \quad (833)$$

30: Modeling Quantum Wells

Radiative Recombination and Gain Coefficients

where the sum is taken over all lasing modes. The stimulated emission coefficient is computed locally at this active vertex and its value is taken at the lasing energy, $\hbar\omega_i$, of mode i . S_i is the photon rate of mode i , solved from the corresponding photon rate equation of mode i , and $|\Psi_i(x, y)|^2$ is the local optical field intensity of mode i at this active vertex. This stimulated recombination rate is entered in the continuity equations to account for the correct depletion of carriers by stimulated emissions.

The total stimulated recombination rates on active vertices can be plotted by including the keyword `StimulatedRecombination` in the `Plot` section of the command file.

Spontaneous Recombination Rate

Spontaneous emissions also deplete the carrier population in the active region and must be taken into consideration. The spontaneous recombination rate at each active vertex is:

$$R_{\text{tot}}^{\text{sp}}(x, y, z) = \int_0^{\infty} r^{\text{sp}}(E) \rho^{\text{opt}}(E) dE \quad (834)$$

where the optical mode density is:

$$\rho^{\text{opt}}(E) = \frac{n_g^2 E^2}{\pi^2 \hbar^3 c^2} \quad (835)$$

The spontaneous recombination rate is an integral over energy space, and this rate is entered into the carrier continuity equations.

The total spontaneous recombination rates on active vertices can be plotted by including the keyword `SpontaneousRecombination` in the `Plot` section of the command file.

Spontaneous Emission Power for LEDs

The total spontaneous emission power density at each active vertex (units of $\text{Js}^{-1} \text{m}^{-3}$) is:

$$\Delta P^{\text{sp}}(x, y, z) = \int_0^{\infty} r^{\text{sp}}(E) \rho^{\text{opt}}(E) \cdot (\hbar\omega) dE \quad (836)$$

where $\rho^{\text{opt}}(E)$ has been defined in Eq. 835. This equation is similar to Eq. 834, except that an additional energy term, $E = \hbar\omega$, is included in the integrand to account for the energy spectrum of the spontaneous emission.

In LED simulations, the spontaneous emission spectrum, $r^{\text{sp}}(E)$, broadens significantly with increasing injected carriers. The integral in [Eq. 834](#) and [Eq. 836](#) are based on a Riemann sum and, as with numeric integration, truncates at an internally set energy value. You can change the truncation value and the Riemann integration interval with the following syntax in the command file:

```
Physics {...  
    LED (...  
        Optics (...)  
        SponScaling = 1.0  
        SponIntegration(<energyspan>, <numpoints>)  
    )  
}
```

where `<energyspan>` is a floating-point number (in eV) and is measured from the edge of the energy bandgap, and `<numpoints>` is an integer denoting the number of discretized intervals to use within this energy span.

The total spontaneous emission power is the volume integral of the power density over all the active vertices:

$$P_{\text{total}}^{\text{sp}} = \int_{(\text{active - region})} \Delta P^{\text{sp}}(x, y, z) dV \quad (837)$$

This is the total spontaneous emission power that is computed and output in an LED simulation.

If the photon-recycling feature is activated, additional powers will be output. For the simple photon-recycling model (see [Simple Photon-Recycling Model \(Active Region\) on page 738](#)), an additional amplified spontaneous emission (ASE) power will appear in the plot (.plt) file. In the full photon-recycling model (see [Full Photon-Recycling Model \(Active Region\) on page 743](#)), there will more power outputs: ASE power, absorption power, and re-emission power.

Fitting Stimulated and Spontaneous Emission Spectra

Where, for performance reasons, full $\mathbf{k} \cdot \mathbf{p}$ and manybody calculations (see [The k.p Method on page 825](#)) are not feasible, it is possible to apply some corrections to the stimulated and spontaneous emission spectra that result from a simplified treatment. The spectra can be scaled in magnitude by the keywords `StimScaling=<float>` and `SponScaling=<float>` in the `Physics-Laser` section. It is also possible to shift the effective emission wavelength (and, therefore, the spectra) by an energy amount specified as `GainShift=<float>`.

Gain-broadening Models

Three different line-shape broadening models are available: Lorentzian, Landsberg, and hyperbolic-cosine. These line-shape functions, $L(E)$, are embedded in the radiative emission coefficients in [Eq. 820](#) and [Eq. 821](#) to account for broadening of the gain spectrum.

Lorentzian Broadening

Lorentzian broadening assumes that the probability of finding an electron or a hole in a given state decays exponentially in time [\[2\]](#). The line-shape function is:

$$L(E) = \frac{\Gamma/(2\pi)}{(E_g - \hbar\omega + E)^2 + (\Gamma/2)^2} \quad (838)$$

Landsberg Broadening

The Landsberg model gives a narrower, asymmetric line-shape broadening, and its line-shape function is:

$$L(E) = \frac{(\Gamma(E))/(2\pi)}{(E_g - \hbar\omega + E)^2 + (\Gamma(E)/2)^2} \quad (839)$$

where:

$$\Gamma(E) = \Gamma \sum_{k=0}^3 a_k \left(\frac{E}{q\Psi_p - q\Psi_n} \right)^k \quad (840)$$

and $q\Psi_p - q\Psi_n$ is the quasi-Fermi level separation. The coefficients a_k are:

$$\begin{aligned} a_0 &= 1 \\ a_1 &= -2.229 \\ a_2 &= 1.458 \\ a_3 &= -0.229 \end{aligned} \quad (841)$$

Hyperbolic-Cosine Broadening

The hyperbolic-cosine function has a broader tail on the low-energy side compared to Lorentzian broadening, and the line-shape function is:

$$L(E) = \frac{1}{4\Gamma} \cdot \frac{1}{\cosh^2\left(\frac{E}{2\Gamma}\right)} \quad (842)$$

Syntax to Activate Broadening

You can select only one line-shape function for gain broadening. This is activated by the keyword `Broadening` in the `Physics-Laser` section of the command file:

```
Physics {...  
  Laser (...  
    Optics (...)  
    # --- Lineshape broadening functions, choose one only ----  
    Broadening (Type=Lorentzian Gamma=0.01)  
    # Broadening (Type=Landsberg Gamma=0.01)      # Gamma in [eV]  
    # Broadening (Type=CosHyper Gamma=0.01)  
  )  
}
```

`Gamma` is the line width Γ , which must be defined in units of eV. If no `Broadening` keyword is detected, Sentaurus Device assumes the gain is unbroadened and does not perform the energy integral in [Eq. 820](#) and [Eq. 821](#).

Nonlinear Gain Saturation Effects

Nonlinear gain saturation is caused by the interaction of increasing light intensity with the optical matrix element, and this effect is important in the study of modulation response. An infinite order perturbation approach is used in a density matrix formulation to derive the nonlinear gain effects [\[3\]](#). Using this approach, the stimulated emission coefficient is derived to be [\[4\]](#):

$$r^{st}(\hbar\omega) = \sum_{i,j} \int dE C_o k_{st} |M_{i,j}|^2 D(E) \times (f_i^C(E) + f_j^V(E) - 1) L(E) \quad (843)$$

30: Modeling Quantum Wells

Nonlinear Gain Saturation Effects

where:

$$L(E) = \frac{\Gamma/(2\pi)}{(E - \hbar\omega)^2 + (\Gamma/2)^2 + \varepsilon S} \quad (844)$$

$$\varepsilon = \frac{2(\tau_c + \tau_v)|M_{i,j}|^2(\hbar\omega)\hbar^2}{\tau_{in}m_0^2\varepsilon_0n_g^2E} \cdot |\Psi(x, y)|^2 \quad (845)$$

τ_c and τ_v are the intraband scattering times of the electrons and holes, respectively. Γ is the line width, which is defined by $\Gamma = 2(\hbar/\tau_{in})$. τ_{in} is the polarization relaxation time and can be defined as:

$$\frac{1}{\tau_{in}} = \frac{1}{2} \left(\frac{1}{\tau_c} + \frac{1}{\tau_v} \right) + \frac{1}{\tau_{sp}} \quad (846)$$

where τ_{sp} is the electron spontaneous emission lifetime and $|\Psi(x, y)|^2$ is the normalized local optical intensity. The stimulated emission coefficient with nonlinear gain effects in Eq. 843 is of the same form as the original stimulated emission coefficient in Eq. 820. The variables in the two coefficients are the same, except for the gain-broadening function, $L(E)$.

The nonlinear gain saturation effect can be included in the form of a broadening function. The simple gain saturation model commonly cited is $g = g_0/(1 + \varepsilon S)$ (see below). This simple form gives a homogeneous broadening and can be approximated from Eq. 843 by assuming $(\hbar/\tau_{in}) \gg (E - \hbar\omega)$. In this model, $|M_{i,j}|^2$ has no spectral dependence by definition.

Since the nonlinear gain saturation effect exists in the form of a broadening function, it can be activated by the keyword Broadening in the Physics-Laser section of the command file:

```
Physics {...  
  Laser (...  
    Optics (...)  
    # ----- Gain saturation model -----  
    Broadening(Type = LorentzianSat  
      Gamma = 4.39e-4          # [eV]  
      eIntrabandRelTime = 1e-13# [s]  
      hIntrabandRelTime = 1e-13# [s]  
      PolarizationRelTime = 3e-12# [s]  
    )  
  )  
}
```

You must set the electron and hole intraband scattering times with `eIntrabandRelTime` and `hIntrabandRelTime`, respectively. The `PolarizationRelTime` can be computed from Eq. 846.

NOTE If the nonlinear gain saturation is activated, you cannot select other gain-broadening functions.

As an alternative to the calculation of gain saturation, Sentaurus Device offers a simple local gain saturation model of the form $g = g_0/(1 + \varepsilon S)$ where you must specify the gain saturation coefficient ε . In contrast to the model described above, this model is not implemented as a broadening model, but as a correction to the coefficient of stimulated emission (see [Eq. 843](#)).

The local epsilon model can be activated in the Physics section:

```
Physics {...  
    Laser (...  
        Optics (...)  
        # ----- Simple gain saturation model -----  
        GainSaturation(  
            Type = LocalEpsilonModel  
            EpsilonSat = 1e-16  
        )  
    )  
}
```

Simple Quantum-Well Subband Model

This section describes the solution of the Schrödinger equation for a simple finite quantum-well model. This is the default model in Sentaurus Device. A more advanced model using the $k \cdot p$ method is available (see [The k.p Method on page 825](#)). This simple quantum-well (QW) subband model is combined with separate QW strain (see [Strain Effects on page 820](#)) and polarization dependence of the optical matrix element (see [Polarization-dependent Optical Matrix Element on page 822](#)) to model most quantum well systems.

In a quantum well, the carriers are confined in one direction. Of interest are the subband energies and wavefunctions of the bound states, which can be solved from the Schrödinger equation. In this simple QW subband model, it is assumed that the bands for the electron, heavy hole, and light hole are decoupled, and the subbands are solved independently by a 1D Schrödinger equation.

The time-independent 1D Schrödinger equation in the effective mass approximation is:

$$\left(-\frac{\hbar^2}{2} \frac{\partial}{\partial x} \frac{1}{m(x)} \frac{\partial}{\partial x} + V(x) - E^i\right) \zeta^i(x) = 0 \quad (847)$$

where $\zeta^i(x)$ is the i -th quantum mechanical wavefunction, E^i is the i -th energy eigenvalue, and $V(x)$ is the finite well shape potential.

30: Modeling Quantum Wells

Simple Quantum-Well Subband Model

With the following ansatz for the even wavefunctions:

$$\zeta(x) = C_1 \begin{cases} \cos\left(\frac{\kappa l}{2}\right) e^{-\alpha(|x|-l/2)} & , |x| > l/2 \\ \cos(\kappa x) & , |x| \leq l/2 \end{cases} \quad (848)$$

and the odd wavefunctions:

$$\zeta(x) = C_2 \begin{cases} \pm \sin\left(\frac{\kappa l}{2}\right) e^{\mp(x \mp l/2)} & , |x| > l/2 \\ \sin(\kappa x) & , |x| \leq l/2 \end{cases} \quad (849)$$

[Eq. 847](#) becomes [2]:

$$\alpha \frac{l}{2} + \frac{m_b}{m_w} \kappa \frac{l}{2} \cot\left(\kappa \frac{l}{2}\right) = 0 \quad (850)$$

$$\alpha \frac{l}{2} - \frac{m_b}{m_w} \kappa \frac{l}{2} \tan\left(\kappa \frac{l}{2}\right) = 0 \quad (851)$$

with:

$$\kappa = \frac{\sqrt{2m_w E}}{\hbar} \quad (852)$$

$$\alpha = \frac{\sqrt{2m_b(\Delta E_c - E)}}{\hbar} \quad (853)$$

The first transcendental equation gives the even eigenvalues, and the second one gives the odd eigenvalues. The wavefunctions are immediately obtained with [Eq. 848](#) and [Eq. 849](#) after the subband energy E has been computed. Having obtained the wavefunctions and subband energies, the carrier densities of the 1D-confined system are also computed by:

$$n(x) = N_e^{2D} \sum_i |\zeta_i(x)|^2 F_0(\eta_n - E_i) \quad (854)$$

$$p(x) = N_{hh}^{2D} \sum_i |\zeta_{j(x)}|^2 F_0(\eta_p - E_{hh}^i) + N_{lh}^{2D} \sum_m |\zeta_{m(x)}|^2 F_0(\eta_p - E_{lh}^m) \quad (855)$$

where $F_0(x)$ is the Fermi integral of the order 0, and η_n , η_p denote the chemical potentials. The indices hh and lh denote the heavy and light holes, respectively.

The effective densities of states are:

$$N_e^{2D} = \frac{k_B T m_e}{\hbar^2 \pi L_x} \quad (856)$$

$$N_{lh/hh}^{2D} = \frac{k_B T m_{lh/hh}}{\hbar^2 \pi L_x} \quad (857)$$

where L_x is the thickness of the quantum well. The thickness of each quantum well is automatically detected in Sentaurus Device by scanning the material regions for the keyword Active.

The effective masses of the carriers in the quantum well can be changed inside the parameter file:

```
eDOSMass
{
    * For effective mass specification Formula1 (me approximation) :
    * or Formula2 (Nc300) can be used :
        Formula = 2      # [1]
    * Formula2:
    * me/m0 = (Nc300/2.540e19)^2/3
    * Nc(T) = Nc300 * (T/300)^3/2
        Nc300 = 8.7200e+16    # [cm-3]
    * Mole fraction dependent model.
    * If just above parameters are specified, then its values will be
    * used for any mole fraction instead of an interpolation below.
    * The linear interpolation is used on interval [0,1].
        Nc300(1)      = 6.4200e+17    # [cm-3]
}
...
SchroedingerParameters:
{ * For the hole masses for Schroedinger equation you can
  * use different formulas.
  * formula=1 (for materials with Si-like hole band structure)
  * m(k)/m0=1/(A+-sqrt(B+C*((xy)^2+(yz)^2+(zx)^2)))
  * where k=(x,y,z) is unit normal vector in reciprocal
  * space. '+' for light hole band, '-' for heavy hole band
  * formula=2: Heavy hole mass mh and light hole mass ml are
  * specified explicitly.
  * Formula 2 parameters:
        Formula = 2      # [1]
        ml       = 0.027 # [1]
        mh       = 0.08  # [1]
  * Mole fraction dependent model.
  * If just above parameters are specified, then its values will be
  * used for any mole fraction instead of an interpolation below.
```

30: Modeling Quantum Wells

Strain Effects

```
* The linear interpolation is used on interval [0,1].  
    ml(1) = 0.094 # [1]  
    mh(1) = 0.08 # [1]  
}
```

Syntax for Simple Quantum-Well Model

This simple QW subband model is the default model when the `QWTransport` and QW scattering model (`QWScatModel`) are activated. [Table 181 on page 1148](#) provides the keywords that are associated with this simple QW model. These keywords are in the command file syntax of [Quantum-Well Scattering Model on page 806](#).

Strain Effects

In semiconductor laser design, it is well known that strain of the quantum well modifies the laser characteristics. Due to the deformation potentials in the crystal at the well–bulk interface and valence band mixing effects, band structure modifications occur mainly for the valence bands. They have an impact on the optical recombination and transport properties. A more advanced $k \cdot p$ model is available, which incorporates strain into the Hamiltonian implicitly (see [The k.p Method on page 825](#)).

In the simple QW subband model discussed in the previous section, a simpler approach to the QW strain effects is adopted. The simple QW subband model does not include nonparabolicities of the band structure, arising from valence band mixing and strain, in a rigorous manner. However, by carefully selecting the effective masses in the well, a good approximation of the strained band structure can be obtained [5]. The effective masses can be changed in the parameter file as previously shown.

Basically, strain has two impacts on the band structure. Due to the deformation potentials, the effective band offsets of the conduction and valence bands are modified. This is included in the simple QW subband model by:

$$\delta E_C = 2a_c \left(1 - \frac{C_{12}}{C_{11}}\right) \varepsilon \quad (858)$$

$$\delta E_V^{HH} = 2a_v \left(1 - \frac{C_{12}}{C_{11}}\right) \varepsilon + b \left(1 + 2 \frac{C_{12}}{C_{11}}\right) \varepsilon \quad (859)$$

$$\delta E_V^{LH} = 2a_v \left(1 - \frac{C_{12}}{C_{11}}\right) \varepsilon - b \left(1 + 2 \frac{C_{12}}{C_{11}}\right) \varepsilon + \frac{\Delta}{2} - \left(-\frac{1}{2} \sqrt{\Delta^2 + 9\delta E_{sh}^2 - 2\delta E_{sh}\Delta}\right) \quad (860)$$

where a_n and a_c are the hydrostatic deformation potential of the conduction and valence bands, respectively. The shear deformation potential is denoted with b and Δ is the spin-orbit split-off energy. The elastic stiffness constants are C_{11} and C_{12} , and ε is the relative lattice constant difference in the active region. The hydrostatic component of the strain shifts the conduction band offset by δE_C and shifts the valence band offset by $\delta E_V^0 = 2a_v(1 - C_{12}/C_{11})\varepsilon$.

The shear component of the strain decouples the light hole and heavy hole bands at the Γ point, and shifts the valence bands by an amount of $\delta E_{sh} = b(1 + 2C_{12}/C_{11})\varepsilon$ in opposite directions.

Syntax for Quantum-Well Strain

The strain shift can be activated by the keyword `Strain` in the `Physics-Laser` section of the command file:

```
Physics {...  
  Laser (...  
    Optics (...)  
    # --- QW physics ---  
    QWTransport  
    QWExtension = AutoDetect  
    QWScatModel  
    QWeScatTime = 8e-13      # [s]  
    QWhScatTime = 4e-13      # [s]  
    eQWMobility = 5370        # [cm^2/Vs]  
    hQWMobility = 150         # [cm^2/Vs]  
    # --- QW strain ---  
    Strain  
  )  
}
```

The parameters a_n , a_c , and b can be entered as `a_nu`, `a_c`, and `b_shear`, respectively, in the `QWStrain` section of the parameter file:

```
QWStrain  
{  
  * Deformation Potentials (a_nu, a_c, b, C_12, C_11  
  * and strainConstant eps :  
  * Formula:  
  * eps = (a_bulk - a_active)/a_active  
  * dE_c = ...  
  * dE_lh = ...  
  * dE_hh = ...  
    eps      = -1.0000e-02# [1]  
  * a_nu   = 1.27  # [1]
```

30: Modeling Quantum Wells

Polarization-dependent Optical Matrix Element

```
* a_c    = -5.0400e+00# [1]
* b_shear = -1.7000e+00# [1]
* C_11 = 10.11   # [1]
* C_12 = 5.61   # [1]
}
```

The elastic stiffness constants C_{11} and C_{12} can be specified by c_{11} and c_{12} . Due to valence band mixing and strain, the valence bands can become nonparabolic. However, within a small range from the band edge, parabolicity can still be assumed. In the simulation, you can modify the effective heavy hole and light hole masses in the parameter file for the subband calculation to account for this effect.

The spin-orbit split-off energy can be specified in the `BandstructureParameters` section of the parameter file:

```
BandstructureParameters{
  ...
  so = 0.34   # [eV]
  ...
}
```

Polarization-dependent Optical Matrix Element

The polarization dependence of the optical matrix element P_{ij} in Eq. 823, p. 809 can be treated in an elegant way for the simple QW subband model. Define the optical polarization angle, γ , as the angle of the vectorial optical field between itself and the quantum well (QW) plane, as shown in Figure 108.

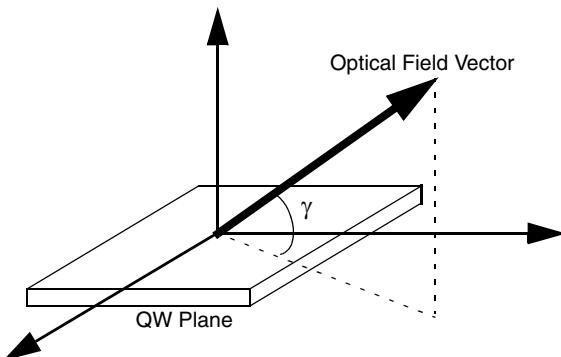


Figure 108 Taking the optical polarization with respect to QW plane so that anisotropic polarization-dependent factor in optical matrix element can be defined

If the vector lies completely in the QW plane as in the case of TE modes, the angle is 0. If it is strictly perpendicular as in TM modes, it is $\pi/2$. The unit is radian.

These polarization angles at each vertex can be visualized by including the keywords `OpticalPolarizationAngleMode0` to `OpticalPolarizationAngleMode9` in the `Plot` section of the command file.

NOTE In bulk material, the optical matrix element is isotropic and the polarization-dependent factor can be taken as $P_{ij} = 1$.

The polarization-dependent factor P_{ij} is a measure of the influence of the polarization of the optical field with the plane wavefunctions of the optical transition. Suppose the optical field polarization is defined by the unit vector:

$$\hat{e} = \alpha \hat{a}_{\text{TE}} + \beta \hat{a}_{\text{TM}} \quad (861)$$

where $\alpha^2 + \beta^2 = 1$. It is obvious that $\alpha = \cos\gamma$ and $\beta = \sin\gamma$ from [Figure 108](#).

The resultant polarization-dependent factor P_{ij} for a quantum well can be derived as:

$$\begin{aligned} P_{c, hh} &= \frac{1}{M_b^2} \left| \hat{e} \bullet \langle iS|\vec{p}| \frac{3}{2}, -\frac{3}{2} \rangle' \right|^2 \\ &= 3 \left\{ \frac{1}{4} \alpha^2 \cos^2 \Psi + \frac{1}{4} \alpha^2 + \frac{1}{2} \beta^2 \sin^2 \Psi \right\} \end{aligned} \quad (862)$$

for C-HH transitions, and:

$$\begin{aligned} P_{c, lh} &= \frac{1}{M_b^2} \left\{ \left| \hat{e} \bullet \langle iS|\vec{p}| \frac{3}{2}, -\frac{1}{2} \rangle' \right|^2 + \left| \hat{e} \bullet \langle iS|\vec{p}| \frac{3}{2}, \frac{1}{2} \rangle' \right|^2 \right\} \\ &= 3 \left\{ \frac{1}{3} \alpha^2 \sin^2 \Psi + \frac{1}{12} \alpha^2 \cos^2 \Psi + \frac{1}{12} \alpha^2 + \frac{2}{3} \beta^2 \cos^2 \Psi + \frac{1}{6} \beta^2 \sin^2 \Psi \right\} \end{aligned} \quad (863)$$

for C-LH transitions.

The **k**-wavevector angle is defined as:

$$\cos^2 \Psi = \frac{E_g + E_i + E_j}{E}, E > E_g + E_i + E_j \quad (864)$$

The cross-coupling terms between the TE and TM contributions evaluate to zero upon integration over the polar angle in the QW plane. This is equivalent to computing different TE-stimulated and TM-stimulated emission coefficients and, therefore, different modal gains for the TE and TM polarizations, which are subsequently summed to give the overall modal gain of the mode at each active vertex, that is:

$$G_{\text{modal}}(x, y) = r_{\text{TE}}^{st} |\bar{E}_{\text{TE}}(x, y)|^2 + r_{\text{TM}}^{st} |\bar{E}_{\text{TM}}(x, y)|^2 \quad (865)$$

30: Modeling Quantum Wells

Polarization-dependent Optical Matrix Element

where $|\bar{E}_{\text{TE}}(x, y)|^2$ and $|\bar{E}_{\text{TM}}(x, y)|^2$ are the local normalized intensities of the vectorial optical field projected onto the TE and TM planes, respectively. In this way, automatic polarization-dependent gain calculations for each separate vertex in the active region are achieved if the vectorial optical solver (FEVectorial) is chosen.

In the case of scalar optical solvers (FEScalar), you can select the type of polarization (TE or TM) in the Physics-Laser-Optics-FEScalar section of the command file:

```
Physics { ...
    Laser (... 
        Optics(
            FEScalar(EquationType = Waveguide
                Symmetry = Symmetric
                LasingWavelength = 800           # [nm]
                TargetEffectiveIndex = (3.4 3.34) # initial guess
                TargetLoss = (10.0 12.0)         # initial guess [1/cm]
                # --- Specify the optical polarization ---
                Polarization = (TE TM)
                ModeNumber = 2
            )
        )
    )
}
```

The TE and TM polarizations give a global optical polarization angle of 0 and $\pi/2$, respectively. The default is TE polarization. For purely TE or TM polarizations, [Eq. 862](#) and [Eq. 863](#) reduce to the values in [Table 112](#).

Table 112 Polarization-dependent factor for purely TE and TM polarizations

	TE polarization	TM polarization
C-HH	$\frac{3}{4}(1 + \cos^2\Psi)$	$\frac{3}{2}\sin^2\Psi$
C-LH	$\frac{5}{4} - \frac{3}{4}\cos^2\Psi$	$\frac{1}{2}(1 + 3\cos^2\Psi)$

A self-consistent simulation of four vectorial modes is performed for an edge-emitting laser to show the polarization-dependent effects on the gain. Figure 109 shows the modal gain plotted (using Inspect) as a function of transition energy for the four different modes.

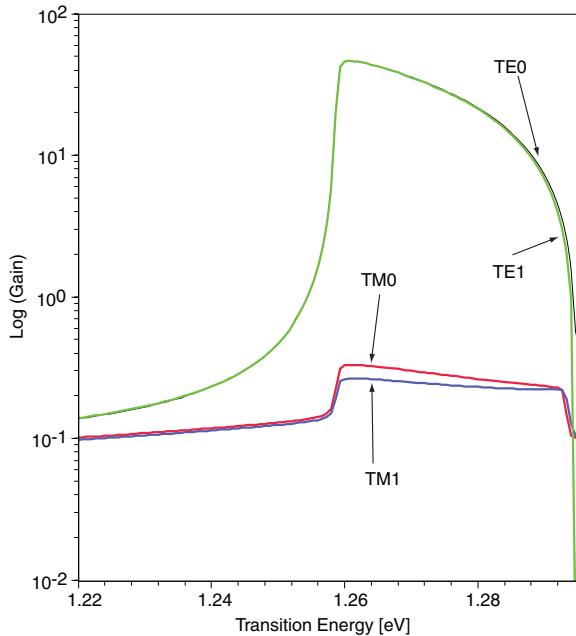


Figure 109 Modal gain curves for two TE and two TM modes

The four modes contain two quasi-TE (TE0 and TE1) and two quasi-TM (TM0 and TM1) modes. The TE gains are much greater than the TM gains because the InGaAs quantum wells used in this example strongly favor TE polarization. Therefore, Sentaurus Device can simulate multimode lasing with mixed optical polarization.

The k.p Method

A more advanced model for computing the QW subbands, strain shift, manybody effects, and so on with the $k \cdot p$ method is available in Sentaurus Device. The $k \cdot p$ method implemented in Sentaurus Device is based on the Luttinger–Kohn model. Choices of 4×4 , 6×6 , and 8×8 Hamiltonians are possible to handle degenerate heavy hole, light hole, spin-orbit split-off, and conduction bands.

The wavefunction in a periodic lattice can be expanded into plane waveforms and is assumed to be a Bloch function:

$$\psi_{nk}(\mathbf{r}) = e^{i\mathbf{k} \cdot \mathbf{r}} \cdot u_{nk}(\mathbf{r}) \quad (866)$$

30: Modeling Quantum Wells

The k.p Method

where $u_{nk}(\mathbf{r})$ is the envelope wavefunction, which is periodic across each crystal lattice, $u_{nk}(\mathbf{r}) = u_{nk}(\mathbf{r} + \mathbf{R})$. Substituting this into the Schrödinger equation, $\mathbf{H}\psi_{nk}(\mathbf{r}) = E_n(\mathbf{k})\psi_{nk}(\mathbf{r})$ gives the eigenvalue equation for the envelope wavefunction:

$$\mathbf{H}u_{nk}(\mathbf{r}) = E(\mathbf{k})u_{nk}(\mathbf{r}) \quad (867)$$

where the Luttinger–Kohn Hamiltonian is:

$$\mathbf{H} = \frac{\mathbf{p}^2}{2m} + V(\mathbf{r}) + \frac{\hbar^2 \mathbf{k}^2}{2m_0} + \frac{\hbar}{4m_0 c^2} \nabla V \times \mathbf{p} \cdot \boldsymbol{\sigma} + \frac{\hbar}{m_0} \mathbf{k} \cdot \mathbf{p} \quad (868)$$

$\boldsymbol{\sigma}$ is the Pauli spin matrix and V is the potential variation of the quantum well region, which may include carrier interaction effects.

In Sentaurus Device, two types of carrier interaction effect are treated:

- Free carrier theory (FCT), which assumes that the carriers do not interact with each other and, therefore, do not contribute to any additional potential terms.
- Manybody effects, which account for the Coulombic interaction between the carriers. This is included in the form of a screened Hartree–Fock (SHF) potential.

When the $\mathbf{k} \cdot \mathbf{p}$ method is used, the valence band mixing effects and manybody effects (if included) distort the band structure from a parabolic shape. In this case, the integral in the stimulated and spontaneous emission coefficients ([Eq. 820, p. 809](#) and [Eq. 821, p. 809](#)) is performed in \mathbf{k} -space rather than energy space. The features of the $\mathbf{k} \cdot \mathbf{p}$ method are:

- Lorentzian and hyperbolic-cosine gain broadening models are available.
- For zinc-blende crystal structure, you can select the 4×4 , 6×6 , and 8×8 $\mathbf{k} \cdot \mathbf{p}$ Hamiltonians with the FCT and SHF carrier interaction effects.
- For wurtzite crystal structure, you can select only the 6×6 $\mathbf{k} \cdot \mathbf{p}$ Hamiltonian with the FCT and SHF carrier interaction effects.
- The screening of piezoelectric fields in GaN-based quantum wells by the carriers contained in the wells can be taken into account (see [Screening Piezoelectric Fields in GaN-type Quantum Wells on page 839](#)).

Luttinger–Kohn Parameters and Hamiltonians for Zinc-Blende Crystal Structure

The Luttinger–Kohn parameters γ_1 , γ_2 , and γ_3 are well known. They are used in the following expressions, which are key components for forming the Luttinger–Kohn Hamiltonian matrices:

$$P = \left(\frac{\hbar^2\gamma_1}{2m}\right)(k_x^2 + k_y^2 + k_z^2) \quad (869)$$

$$Q = \left(\frac{\hbar^2\gamma_2}{2m}\right)(k_x^2 + k_y^2 - 2k_z^2) \quad (870)$$

$$R = -\left(\frac{\hbar^2\gamma_2}{2m}\right)\sqrt{3}(k_x^2 - k_y^2) + i\left(\frac{\hbar^2\gamma_3}{2m}\right)2\sqrt{3}k_xk_y \quad (871)$$

$$S = \left(\frac{\hbar^2\gamma_3}{2m}\right)2\sqrt{3}(k_x - ik_y)k_z \quad (872)$$

The strain-related parameters are:

$$\varepsilon_{xx} = \varepsilon_{yy} = \frac{a_{0b} - a_{0w}}{a_{0w}} \quad (873)$$

$$\varepsilon_{zz} = -2\frac{C_{12}}{C_{11}}\varepsilon_{xx} \quad (874)$$

$$P_\varepsilon = a_v(\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz}) = -\delta E_v \quad (875)$$

$$Q_\varepsilon = -\frac{b}{2}(\varepsilon_{xx} + \varepsilon_{yy} - 2\varepsilon_{zz}) \quad (876)$$

$$\delta E_c = 2a_c\left(1 - \frac{C_{12}}{C_{11}}\right)\varepsilon_{xx} \quad (877)$$

where a_{0w} and a_{0b} are the lattice constants of the well and barrier, respectively. C_{11} , C_{12} , a_v , and b are deformation potentials. As shorthand notations, the following are defined:

$$P_t = P + P_\varepsilon \quad (878)$$

$$Q_t = Q + Q_\varepsilon \quad (879)$$

30: Modeling Quantum Wells

The k.p Method

The envelope wavefunction, u_{nk} , is also expanded to a set of basis functions: p-like for the valence bands and s-like for the conduction bands. Valence band mixing is also included in this expansion.

Four-Band Hamiltonian

The 4×4 Hamiltonian gives the solution to degenerate heavy-hole and light-hole bands; the conduction band is treated independently. Its matrix form is:

$$\mathbf{H} = \begin{bmatrix} P_t + Q_t & -S & R & 0 \\ -S^* & P_t - Q_t & 0 & R \\ R^* & 0 & P_t - Q_t & S \\ 0 & R^* & S^* & P_t + Q_t \end{bmatrix} \quad (880)$$

where * denotes complex conjugate.

Six-Band Hamiltonian

The 6×6 Hamiltonian gives the solution to degenerate heavy-hole, light-hole, and split-off bands; the conduction band is treated independently. Its matrix form is:

$$\mathbf{H} = \begin{bmatrix} P_t + Q_t & R & \sqrt{2}R & -\frac{S}{\sqrt{2}} & -S & 0 \\ R^* & P_t - Q_t & \sqrt{2}Q & \sqrt{\frac{3}{2}}S^* & 0 & S \\ \sqrt{2}R^* & \sqrt{2}Q & P_t + \Delta & 0 & \sqrt{\frac{3}{2}}S^* & -\frac{S}{\sqrt{2}} \\ -\frac{S}{\sqrt{2}} & \sqrt{\frac{3}{2}}S & 0 & P_t + \Delta & -\sqrt{2}Q & -\sqrt{2}R \\ -S^* & 0 & \sqrt{\frac{3}{2}}S & -\sqrt{2}Q & P_t - Q_t & R \\ 0 & S^* & -\frac{S}{\sqrt{2}} & -\sqrt{2}R^* & R^* & P_t + Q_t \end{bmatrix} \quad (881)$$

where Δ is the spin-orbit split-off energy.

Eight-Band Hamiltonian

The 8×8 Hamiltonian gives the solution to degenerate heavy-hole, light-hole, split-off, and conduction bands. Its matrix form is:

$$H = \begin{bmatrix} E_g + s \frac{\hbar^2 k^2}{2m_0} + \delta E_c & -2P_z k_z & P_z k_z & \sqrt{\frac{3}{2}} P_+ & 0 & -\frac{P_-}{\sqrt{2}} & -P_- & 0 \\ -\sqrt{2} P_z^* k_z & -(P_t - Q_t) & -\sqrt{2} Q_t & S^* & -\frac{P_+^*}{\sqrt{2}} & 0 & -\frac{\sqrt{3}}{2} S & R \\ P_z^* k_z & -\sqrt{2} Q_t & -(P_t + \Delta) & -\frac{S^*}{\sqrt{2}} & -P_+^* & \sqrt{\frac{3}{2}} S & 0 & \sqrt{2} R \\ \sqrt{\frac{3}{2}} P_+^* & S & -\frac{S}{\sqrt{2}} & -(P_t + Q_t) & 0 & -R & -\sqrt{2} R & 0 \\ 0 & -\frac{P_+}{\sqrt{2}} & -P_+ & 0 & E_g + s \frac{\hbar^2 k^2}{2m_0} + \delta E_c & \sqrt{2} P_z^* k_z & -P_z k_z & -\frac{\sqrt{3}}{2} P_- \\ -\frac{P_-^*}{\sqrt{2}} & 0 & \sqrt{\frac{3}{2}} S^* & -R^* & \sqrt{2} P_z^* k_z & -(P_t - Q_t) & -\sqrt{2} Q_t & S \\ -P_-^* & -\sqrt{\frac{3}{2}} S^* & 0 & -\sqrt{2} R^* & -P_z^* k_z & -\sqrt{2} Q_t & -(P_t - \Delta) & -\frac{S}{\sqrt{2}} \\ 0 & R^* & \sqrt{2} R^* & 0 & -\sqrt{\frac{3}{2}} P_-^* & S^* & -\frac{S}{\sqrt{2}} & -(P_t + Q_t) \end{bmatrix} \quad (882)$$

where:

$$P_z = i \frac{P}{\sqrt{3}} \quad (883)$$

$$P_{\pm} = i \frac{P}{\sqrt{3}} (k_x \pm k_y) \quad (884)$$

and s is a dimensionless parameter for describing the effect of the free-electron term in the Kane approach [6].

Luttinger–Kohn Parameters and Hamiltonian for Wurtzite Crystal Structure

Six-Band Hamiltonian

The 6×6 Hamiltonian gives the solution to degenerate heavy-hole, light-hole, and split-off bands; the conduction band is treated independently. The formula from the paper by Chuang and Chang is followed [7].

Its block-diagonalized matrix form is:

$$\mathbf{H} = \begin{bmatrix} F & K_t & -iH_t & 0 & 0 & 0 \\ K_t & G & \Delta - iH_t & 0 & 0 & 0 \\ iH_t & \Delta + iH_t & \lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & F & K_t & iH_t \\ 0 & 0 & 0 & K_t & G & \Delta + iH_t \\ 0 & 0 & 0 & -iH_t & \Delta - iH_t & \lambda \end{bmatrix} \quad (885)$$

where:

$$F = \Delta_1 + \Delta_2 + \lambda + \theta \quad (886)$$

$$F = \Delta_1 - \Delta_2 + \lambda + \theta \quad (887)$$

$$\lambda = \frac{\hbar^2}{2m_0}(A_1k_z^2 + A_2k_t^2) + \lambda_\varepsilon \quad (888)$$

$$\lambda_\varepsilon = D_1\varepsilon_{zz} + D_2(\varepsilon_{xx} + \varepsilon_{yy}) \quad (889)$$

$$\theta = \frac{\hbar^2}{2m_0}(A_3k_z^2 + A_4k_t^2) + \theta_\varepsilon \quad (890)$$

$$\theta_\varepsilon = D_3\varepsilon_{zz} + D_4(\varepsilon_{xx} + \varepsilon_{yy}) \quad (891)$$

$$K_t = \frac{\hbar^2}{2m_0}A_5k_t^2 \quad (892)$$

$$H_t = \frac{\hbar^2}{2m_0}A_6k_zk_t \quad (893)$$

Syntax for k.p Method

The syntax to activate the $k \cdot p$ method is complicated. It involves:

- The definition of a nonlocal mesh across the quantum wells so that a 1D (spatially) Schrödinger equation can be formulated with the $k \cdot p$ method.
- Using new keywords in the Physics-Laser and Solve sections.
- Inputting new coefficients in the QWstrain section of the parameter file.

Constructing Nonlocal Mesh on Straight Line

You must include an additional layer between the separate confinement heterostructure (SCH) and quantum well regions, each on the n and p side of the laser diode, as shown in [Figure 110](#). These additional layers together with the quantum well region define the spatial span where the 1D Schrödinger equation will be solved numerically using the $k \cdot p$ method. The 1D Schrödinger equation is solved by discretizing the line (nonlocal mesh) that traverses these additional layers and the quantum well region in the vertical direction. You can select the discretization size in the command file.

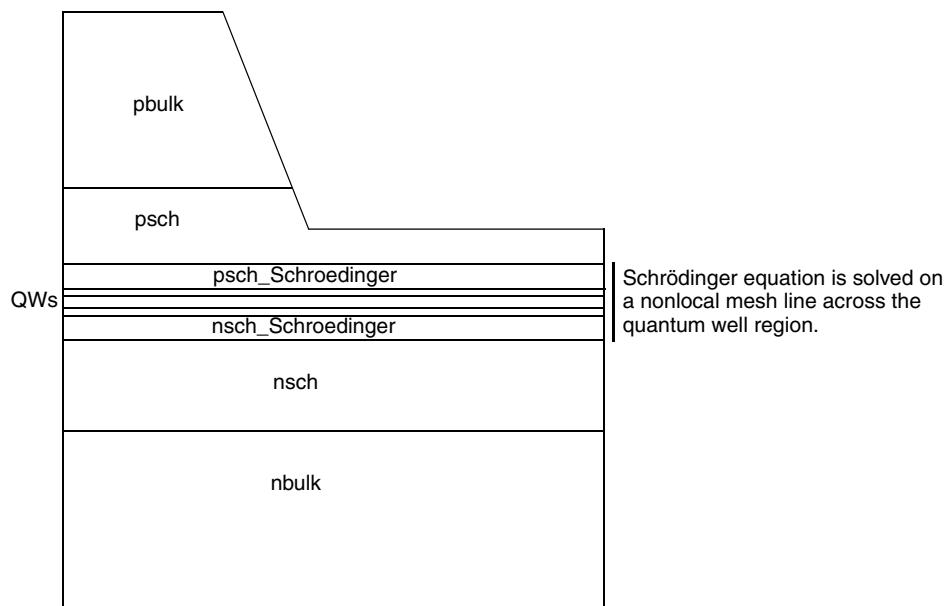


Figure 110 Adding additional layers between the SCH and quantum well regions; these layers are labelled psch_Schroedinger and nsch_Schroedinger as they are associated with the solution of 1D Schrödinger equation

30: Modeling Quantum Wells

The k.p Method

The $k \cdot p$ method is activated by the keyword Bandstructure in the Physics-Laser and Solve sections of the command file:

```
Electrode {
    {Name="p_Contact" voltage=0.9 AreaFactor = 500}
    {Name="n_Contact" voltage=0.0 AreaFactor = 500}
}

File {
    Grid = "mesh_mdr.grd"
    Doping = "mesh_mdr.dat"
    Parameters = "des_las.par"

    Current = "testkp_current"
    Output = "testkp_log"
    Plot = "testkp_plot"
    ModeGain= "testkp_gain"
}

Plot {
    # Include desired variables to plot
}

Physics {
    AreaFactor = 2          # for symmetric devices
    Laser (
        Optics(
            # --- Both scalar and vectorial solvers possible with k.p method ---
            FEScalar( Symmetry = Symmetric
                Polarization = TE           # or TM
                LasingWavelength = 980      # [nm]
                TargetEffectiveIndex = 3.5   # initial guess
                ModeNumber = 1
            )
        )
        TransverseModes             # Edge emitter
        CavityLength = 500          # [micrometers]
        lFacetReflectivity = 0.3
        rFacetReflectivity = 0.3
        OpticalLoss = 10.0          # [1/cm]

        # --- Define basic QW physics and three-point model ---
        QWTransport
        QEExtension = AutoDetect
        QWScatmodel
        QWeScatTime = 1e-13         # [s]
        QWhScatTime = 2e-14         # [s]
        eQWMobility = 9200          # [cm^2/Vs]
        hQWMobility = 400           # [cm^2/Vs]
    )
}
```

```

# --- Activate k.p method with strain effects ---
ManyBodyEffects (Type=FCT) # or SHF
Strain (RefLatticeConst = 5.65392e-10) # [m]
Bandstructure (
    CrystalType = Zincblende
    Order = kp4x4 # or Nokp or kp6x6 or kp8x8
)
# --- Only Lorentzian or CosHyper gain broadening with k.p method ---
Broadening(Type=Lorentzian Gamma=0.013)

)
Mobility (DopingDependence)
Recombination (SRH Auger)
EffectiveIntrinsicDensity (NoBandGapNarrowing)
Fermi
HeteroInterface
Thermionic
}

# --- Define material region physics ---
Physics (region="pbulk") { MoleFraction(xfraction = 0.28) }
Physics (region="nbulk") { MoleFraction(xfraction = 0.28) }
Physics (region="psch") { MoleFraction(xfraction = 0.09) }
Physics (region="nsch") { MoleFraction(xfraction = 0.09) }
Physics (region="psch_Schroedinger") { MoleFraction(xfraction = 0.09) }
Physics (region="nsch_Schroedinger") { MoleFraction(xfraction = 0.09) }
Physics (region="barrl") { MoleFraction(xfraction = 0.09) }

# --- Define the QWs as the active region ---
Physics (region="qw1") {
    MoleFraction( xfraction = 0.8 Grading = 0.00 )
    Active
}
Physics (region="qw2") {
    MoleFraction( xfraction = 0.8 Grading = 0.00 )
    Active
}

# --- Activate Schroedinger solver on non-local mesh defined below ---
Physics (RegionInterface="psch/psch_Schroedinger") {
    Schroedinger (electron maxSolutions(electron)=3
                  hole maxSolutions(hole)=6
                )
}
# --- Define non-local mesh ---
Math (RegionInterface="psch/psch_Schroedinger") {

```

30: Modeling Quantum Wells

The k.p Method

```
Nonlocal( Length = 0.143e-4           # [cm]
          Direction = 2                 # y-direction
          Discretization = 0.2e-7      # [cm]
          )
}
Math (RegionInterface="nsch/nsch_Schroedinger") {
    NonLocal(AnchorPoints)
}

# --- Define regions to be excluded from non-local mesh ---
Math (Region="nsch") {
    NonLocal(-Transparent)
}
Math (Region="psch") {
    NonLocal(-Transparent)
}
Math (Region="pbulk") {
    NonLocal(-Transparent)
}

# ----- Choice and control of numerical methods -----
Math {...}

# ----- Parameters for gain plots -----
GainPlot { range=(1.0 1.2) intervals=60 } # energy range in [eV] and number of
points

Solve {
    Poisson
    Coupled {Electron Hole Poisson QWeScatter QWhScatter}
    Coupled {Electron Hole Poisson QWeScatter QWhScatter PhotonRate}

    # --- Use simple QW subband model first to solve ---
    quasistationary (
        InitialStep=0.05
        MaxStep=0.1
        Minstep=0.00001
        Goal {name="p_Contact" voltage=1.2})
    {
        Plugin(BreakOnFailure) {
            Wavelength
            Coupled {Electron Hole Poisson QWeScatter QWhScatter
                      PhotonRate}
        }
    }

    # --- Then turn on the k.p method ---
    Bandstructure
```

```

Wavelength

# --- Continue using the k.p method in a Gummel iteration for self-
consistency ---
quasistationary (
    InitialStep=0.05
    MaxStep=0.1
    Minstep=0.00001
    # ---
    Plot {range=(0,1) intervals=5}
    PlotGain {range=(0,1) intervals=2}
    Goal {name="p_Contact" voltage=2.5}
{
    Plugin(BreakOnFailure) {
        Bandstructure
        Wavelength
        Coupled{Electron Hole Poisson QWeScatter QWhScatter
            PhotonRate}
    }
}
}

```

The material regions defined in this example correspond to those in [Figure 110 on page 831](#). Some comments about this example are:

- In the Physics-Laser section, the $k \cdot p$ method with various effects is activated by the keywords `Bandstructure`, `Strain`, and `ManyBodyEffects`.
- In the Physics-Laser-Strain section, a reference lattice constant must be specified by the argument `RefLatticeConstant=<float>` to compute the strain parameters.
- In the Physics-Laser-ManyBodyEffects section, two types of manybody effect are available: free carrier theory (keyword `FCT`) and screened Hartree–Fock (keyword `SHF`). The default is the free carrier theory.
- In the Physics-Schroedinger section, the 1D Schrödinger equation is activated to be solved on a nonlocal mesh defined by the boundaries of the `psch_Schroedinger` and `nsch_Schroedinger` layers (see [Figure 110](#)).
- The nonlocal mesh for the 1D Schrödinger equation is defined in the Math-Nonlocal sections. The `AnchorPoints` define the starting location of the 1D nonlocal mesh. The length and discretization size of the mesh can be defined by the keywords `Length` and `Discretization`, respectively. The mesh should only be within the `psch_Schroedinger`, `nsch_Schroedinger`, and `QW` regions only. Therefore, the value that is specified by the keyword `Length` should be approximately equal to the distance between the two interfaces, for which `Length` and `AnchorPoints` have been specified. If the length of the nonlocal mesh exceeds the boundaries, you can manually exclude the unwanted material regions from the nonlocal mesh by using the keyword `NonLocal(-Transparent)`.

30: Modeling Quantum Wells

The k.p Method

- In the Solve section, the simple QW subband model is solved up to near the lasing threshold. After that, the $k \cdot p$ method is included in the Gummel iterations for self-consistent solutions of the system.

Adjusting k.p Parameters in Parameter File

The deformation potentials and other strain parameters required for the $k \cdot p$ method can be defined in the QWStrain section of the parameter file:

```
QWStrain
{
    * Deformation Potentials (a_nu, a_c, b, C_12, C_11).
    * StrainConstant eps (formula = 1) or lattice constant
    * a0 (formula = 2) for energy shift of quantum-well
    * subbands.
    * Formula 1:
    * eps = (a_bulk - a_active)/a_active
    * Formula 2:
    * a0(T) = a0 + alpha (T-Tpar)
    * eps = (a_ref - a0(T))/a_ref

    * dE_c = 2 a_c (1- C12/C11) eps
    * dE_lh = 2 a_nu (1- C12/C11) eps - b (1+ 2 C12/C11) eps
    * dE_hh = 2 a_nu (1- C12/C11) eps + b (1+ 2 C12/C11) eps

    Formula = 2      # [1]

    * Mole fraction dependent model.
    * If only constant parameters are specified, those values will be
    * used for any mole fraction instead of the interpolation below.
    * Linear interpolation is used on the interval [0,1].
        * a_nu(0) = 1          # [eV]
        * a_nu(1) = 1.16       # [eV]
        * a_c(0) = -5.0800e+00 # [eV]
        * a_c(1) = -7.1700e+00 # [eV]
        * b(0) = -1.8000e+00 # [eV]
        * b(1) = -1.7000e+00 # [eV]
        * C_11(0) = 8.329      # [eV]
        * C_11(1) = 11.879      # [eV]
        * C_12(0) = 4.526      # [eV]
        * C_12(1) = 5.376      # [eV]
        * eps(0) = 0.0000e+00 # [1]
        * eps(1) = 0.0000e+00 # [1]
        * a0(0) = 6.059e-10 # [m]
        * a0(1) = 5.654e-10 # [m]
```

```

* alpha(0) = 2.7400e-15 # [m/K]
* alpha(1) = 3.8801e-15 # [m/K]
}

```

For the $k \cdot p$ method, `Formula = 2` must be used in the parameter file and a reference lattice constant (for example, of the substrate) must be specified by the argument `RefLatticeConst=<float>` in the `Physics-Laser-Strain` section of the command file. For non- $k \cdot p$ simulations, the parameter `Eps` in the parameter file is used for strain correction. It is possible to take into account the change of the lattice constant with temperature for $k \cdot p$ and formula 2 only.

The $k \cdot p$ parameters can be defined in the `BandstructureParameters` section of the parameter file:

```

BandstructureParameters
{
    * Parameters for k.p bandstructure calculation:

    * Zincblende crystals:
    * Luttinger parameters gamma_1, gamma_2, gamma_3
    * Spin-orbit split-off energy so
    * Matrix element parameters for TE and TM modes ep_te and ep_tm

    * Wurtzite crystals:
    * Effective mass parameters A1, A2, A3, A4, A5, A6
    * Spin-orbit split-off energy so
    * Crystal-field split energy cr
    * Matrix element parameters for TE and TM modes ep_te and ep_tm
    *
    *

    gamma_1 = 6.85          # [1]
    gamma_2 = 2.1           # [1]
    gamma_3 = 2.9           # [1]
    so     = 0.014          # [eV]
    ep_te  = 18.8           # [eV]
    ep_tm  = 12.4           # [eV]
    cr     = 0.019          # [eV]
    A1     = -7.2400e+00 # [1]
    A2     = -5.1000e-01 # [1]
    A3     = 6.73            # [1]
    A4     = -3.3600e+00 # [1]
    A5     = -3.3500e+00 # [1]
    A6     = -4.7200e+00 # [1]
}

```

30: Modeling Quantum Wells

The k·p Method

The $k \cdot p$ keywords in the command file are summarized in [Table 182 on page 1150](#) and [Table 184 on page 1151](#).

NOTE If the wavelength search algorithm fails due to the manybody simulation or a strong strain shift of the QW subbands, the keyword `NewWavelengthSearch(InitialWavelength=<float>)` can be activated in the `Math` section. This switches on a more robust algorithm that, in most cases, resolves the convergence problem. The specification of an initial wavelength for the new search algorithm is optional. If specified, it must be given in units of nanometers and it should be larger than the expected actual transition wavelength.

Plotting the Local Band Structure Data

The band structures computed by the $k \cdot p$ method can be plotted in a similar fashion as the far field and gain plots. The syntax for this is:

```
File{...
    Bandstructure = "bandfile"
    ...
}

BandstructurePlot{ Vertex=(742 819 1150) }

Solve{...
    Quasistationary (
        InitialStep=0.05
        MaxStep=0.05
        Minstep=0.0003
        Plot {range=(0,1) intervals=3}
        PlotBandstructure {range=(0,1) intervals=3}
        Goal {name="p_Contact" voltage=1.8})
    {
        Plugin (BreakOnFailure Iterations=8) {
            Bandstructure
            Wavelength
            Coupled{Electron Hole Poisson QWeScatter QWhScatter PhotonRate}
        }
    }
    ...
}
```

The activation syntax is similar to that of OptFarField (see [Far Field on page 786](#)) and SaveOptField (see [Saving and Loading Optical Modes on page 783](#)). The highlights of the syntax are:

- In the File section, band structure plotting is activated by the keyword Bandstructure, and the value assigned to it, "bandfile" in this case, becomes the base name for the output files of the band structures.
- In the Solve-Quasistationary section, the argument range=(0,1) in the PlotBandstructure keyword is mapped to the initial and final bias conditions. In this example, the initial and final (goal) p_Contact voltages are 0 V and 1.8 V, respectively. The number of intervals is intervals=3, which gives a total of four (= 3+1) modal gain plots at 0, 0.6, 1.2, and 1.8 V. In general, specifying intervals=n will produce (n+1) plots.
- The section BandstructurePlot allows you to choose the active vertices with the keyword Vertex=(...) from which the band structure will be plotted. These vertices have to be active vertices in the QW three-point model. For each vertex specified, the band structure $E(k)$ is plotted for the different subbands as well as the wavefunctions $\Phi(x)$ on the nonlocal line crossing the vertex. The entries in Vertex=(...) are vertex indices of Sentaurus Device and they can be obtained from Tecplot SV. Set the environment variable TEC_GRID_DEBUGGING, run Tecplot SV, and select **Plot > Label Points and Cells**. In the **Label Points and Cells** dialog box, select the **Show Cell Labels** option, and then from the **Show Variable Value** box, select **VertexIndex**.
- The units in the plot files must be [1/m] and [eV] for the $E(k)$ plot, and [m] and [$1/\sqrt{m}$] for the $\Phi(x)$ plot.
- The output files in this example are:
 - bandfile_kpbandstruc_vertexXXX_000000_des.plt, ..., and
bandfile_kpeigenfunc_vertexXXX_000000_des.plt, ...

These files can be viewed using Inspect.

Screening Piezoelectric Fields in GaN-type Quantum Wells

GaN, AlN, InN, and their alloys have a wurtzite crystal structure, which is non-centrosymmetric with a polar axis along the c-axis. Therefore, in these materials, the misfit strain in heterostructures grown along the c-axis can cause an electric moment due to the piezoelectric effect [8]. In addition, nitride materials exhibit very large spontaneous polarization (that is, polarization at zero strain) parallel to the [0001]-direction [9]. Strain-induced and spontaneous polarization cause interface charges at heterointerfaces between different nitride materials.

30: Modeling Quantum Wells

Optical Emission Including Manybody Effects

These charges can be taken into account in quantum-well laser and LED simulations by specifying the space charge values in the `Physics` sections of the interfaces:

```
Physics (RegionInterface="region1/region2") { Charge(Conc=5e12) }
# Sheet space charge in [cm^-2]
```

At quantum-well interfaces, these space charges lead to a distortion of the conduction band and the valence band and, therefore, to a spatial separation of the centers of electron and hole wavefunctions. This separation, in turn, constitutes a polarization that leads to an electric field counteracting the space charges at the quantum-well interfaces. To simulate the characteristics of a nitride-based device correctly, the screening of the piezoelectric space charges by the carriers in a quantum well must be included in the simulation self-consistently.

This can be performed by specifying the keyword `Screening` as an argument of the `Bandstructure` keyword:

```
Bandstructure(
    CrystalType = Wurtzite
    Order = kp6x6
    Screening
)
```

NOTE Due to the additional feedback from the band structure calculation to the electro-optothermal system of equations, the number of `Plugin` iterations required to achieve self-consistency may increase considerably if `Screening` is switched on.

Optical Emission Including Manybody Effects

The accurate computation of optical emission processes is crucial in active optoelectronic device simulation. This is especially true when electron and hole densities become high. In this case, the Coulombic interaction between carriers modifies the optical and electrical properties of the material. Therefore, Sentaurus Device offers to compute stimulated and spontaneous emission rates including manybody effects.

Physical Model

For computing the band structure, it is sufficient to use a one-electron Hamiltonian that does not contain any carrier-carrier interaction terms (see [The k.p Method on page 825](#)). However, the accurate calculation of optical emission often requires a more rigorous inclusion of manybody effects. The framework in Sentaurus Device is based on a semiclassical theory where carriers are described quantum-mechanically, while the light is treated classically according to Maxwell equations.

Furthermore, the adiabatic approximation is assumed. According to the literature [10], the Hamiltonian including carrier–carrier and carrier–light interaction can be written as:

$$\begin{aligned} H' = & \sum_k \{ [E_g + E_e(\mathbf{k})] a_{\mathbf{k}}^\dagger a_{\mathbf{k}} + E_h(\mathbf{k}) b_{-\mathbf{k}}^\dagger b_{-\mathbf{k}} \} \\ & + \frac{1}{2} \sum_{\mathbf{k}\mathbf{k}' \neq 0} \sum_q [V_q (a_{\mathbf{k}+\mathbf{q}}^\dagger a_{\mathbf{k}-\mathbf{q}} a_{\mathbf{k}'} a_{\mathbf{k}'} + b_{\mathbf{k}+\mathbf{q}}^\dagger b_{\mathbf{k}'-\mathbf{q}} b_{\mathbf{k}'} b_{\mathbf{k}}) - 2 a_{\mathbf{k}+\mathbf{q}}^\dagger b_{\mathbf{k}'-\mathbf{q}}^\dagger b_{\mathbf{k}'} a_{\mathbf{k}}] \\ & - (\mu_{\mathbf{k}} a_{\mathbf{k}}^\dagger b_{-\mathbf{k}}^\dagger + \mu_{\mathbf{k}}^* b_{-\mathbf{k}}^\dagger a_{\mathbf{k}}^\dagger) E(z, t) \end{aligned} \quad (894)$$

where E_g is the un-renormalized material bandgap energy, and $E_e(\mathbf{k})$ and $E_h(\mathbf{k})$ describe the electron and hole kinetic energies given by:

$$\begin{aligned} E_e(\mathbf{k}) &= \frac{\hbar^2 \mathbf{k}^2}{2m_e} \\ E_h(\mathbf{k}) &= \frac{\hbar^2 \mathbf{k}^2}{2m_h} \end{aligned} \quad (895)$$

where a^\dagger , a , b^\dagger , and b are the creation and annihilation operators for electrons and holes. V_q and $E(z, t)$ denote the Fourier-transformed Coulombic potential and the electric field amplitude of the light. $\mu_{\mathbf{k}}$ is the dipole matrix element.

In Eq. 894, the first line describes the energies of the charge carriers. The second line incorporates the Coulombic effects: interband carrier–carrier and electron–hole interactions. The third line describes the carrier–light interaction.

According to [11], the polarization can be written as:

$$P'(z, t) = \frac{1}{V} \sum_{\mathbf{k}} \mu_{\mathbf{k}}^* p_{\mathbf{k}} \quad (896)$$

where V denotes the volume of the active region and $p_{\mathbf{k}}$ is the microscopic polarization. $p_{\mathbf{k}}$ is obtained by solving the Heisenberg equation of motion $p_{\mathbf{k}} = \langle b_{-\mathbf{k}} a_{\mathbf{k}} \rangle$. The general equation of motion for the microscopic polarization reads [10]:

$$\begin{aligned} \frac{dp_{\mathbf{k}}}{dt} &= \frac{d}{dt} \langle b_{-\mathbf{k}} a_{\mathbf{k}} \rangle \\ &= -i\omega_{\mathbf{k}} p_{\mathbf{k}} - \frac{i}{\hbar} \mu_{\mathbf{k}} E(z, t) (n_{e\mathbf{k}} + n_{h\mathbf{k}} - 1) \\ &\quad + \frac{i}{\hbar} \sum_{\mathbf{k}', \mathbf{q} \neq 0} V_q (\langle a_{\mathbf{k}'+\mathbf{q}}^\dagger b_{-\mathbf{k}} a_{\mathbf{k}'} a_{\mathbf{k}'+\mathbf{q}} \rangle + \langle b_{\mathbf{k}'-\mathbf{q}}^\dagger b_{\mathbf{k}'} a_{\mathbf{k}} b_{-\mathbf{k}-\mathbf{q}} \rangle \\ &\quad - \langle a_{\mathbf{k}'+\mathbf{q}}^\dagger b_{-\mathbf{k}} a_{\mathbf{k}'} a_{\mathbf{k}} \rangle - \langle b_{\mathbf{k}'-\mathbf{q}}^\dagger b_{-\mathbf{k}} b_{\mathbf{k}'} a_{\mathbf{k}-\mathbf{q}} \rangle + \langle b_{-\mathbf{k}+\mathbf{q}}^\dagger a_{\mathbf{k}-\mathbf{q}} \rangle \delta_{\mathbf{k}, \mathbf{k}'}) \end{aligned} \quad (897)$$

30: Modeling Quantum Wells

Optical Emission Including Manybody Effects

where ω_k is the transition frequency. In Sentaurus Device, a quasi-equilibrium system is assumed. Consequently, the distribution functions for electron and holes (n_{ek} and n_{hk}) become Fermi–Dirac functions.

The classical expression for the polarization reads:

$$\begin{aligned} P(z, t) &= \frac{1}{2} P(z) e^{i(kz - \omega t - \phi(z))} \hat{e} + cc \\ P(z) &= n_{ref}^2 \epsilon_0 \chi(z) E(z) \end{aligned} \quad (898)$$

where $\chi(z)$ and $E(z)$ denote the complex susceptibility and the electric field, respectively. n_{ref} is the refractive index and ϕ_{ref} describes the phase shift.

The amplitude gain g' is defined as:

$$\frac{dE(z)}{dz} = g' \cdot E(z) = -\frac{\omega n_{ref}}{2c_0} \chi''(z) E(z) \quad (899)$$

where $\chi''(z)$ denotes the imaginary part of the complex susceptibility and c_0 is the speed of light.

[Eq. 898](#) and [Eq. 899](#) are used to obtain the equation for the material gain $g = 2g'$ depending on the microscopic polarization p_k [12]:

$$g(\omega) = \text{Im} \left\{ \frac{-2\omega e^{i(kz - \omega t - \phi(z))}}{E(z) \epsilon_0 n_{ref} c} \cdot \frac{1}{V} \sum_k \mu_k * p_k \right\} \quad (900)$$

The general expressions for the stimulated and spontaneous emission including manybody effects are derived in [12]:

$$\begin{aligned} r^{\text{st}}(\hbar\omega_v) &= \text{Im} \left\{ \frac{i\omega_v}{\hbar c \epsilon_0 n_{ref}^2 \pi L_z} \right. \\ &\quad \left. \cdot \sum_{i,j} \sum_{\sigma} \int_0^{\infty} k_t \left| \mu_{\sigma}^{ij} \right|^2 \frac{(f_i^C(k) + f_j^V(k) - 1)}{i(\omega'(k) - \omega_v) + \gamma(k)} Q_{ij}(k, \omega_v) dk_t \right\} \end{aligned} \quad (901)$$

$$r^{\text{sp}}(\hbar\omega_v) = \text{Im} \left\{ \frac{i\omega_v}{\hbar c \epsilon_0 n_{\text{ref}} 2\pi L_z} \cdot \sum_{i,j} \sum_{\sigma=0}^{\infty} \int dk_t \left| \mu_{\sigma}^{ij}(k) \right|^2 \frac{f_i^C(k) f_j^V(k)}{i(\omega'(k) - \omega_v) + \gamma(k)} Q_{ij}(k, \omega_v) dk_t \right\} \quad (902)$$

where ω_v is the frequency of the lasing mode v , L_z is the width of the quantum well, and k_t is the transverse wavevector. The indices i and j denote the electron and hole subbands, and σ describes the two possible electron spins. $\mu_{\sigma}^{ij}(k)$ is the momentum matrix element. f_i^C and f_j^V are the Fermi–Dirac distributions. $\omega'(k)$, $\gamma(k)$, and $Q_{ij}(k, \omega_v)$ denote the renormalized transition frequency, the dephasing rate, and the Coulombic enhancement factor, respectively.

For tabulated emission computation, three different levels of manybody approximation are available. Depending on the approximation, $\omega'(k)$, $\gamma(k)$, and $Q_{ij}(k, \omega_v)$ are computed differently.

Free Carrier Theory

The carrier are assumed to move without interacting with other particles in the system. Therefore, the terms describing the interband carrier–carrier and electron–hole interactions are neglected in the Hamiltonian. Furthermore, the Coulombic enhancement factor in Eq. 901 and Eq. 902 is neglected ($Q_{ij} = 1$). The dephasing rate γ is entered using phenomenological broadening. The transition frequency $\omega'(k)$ is given by the un-renormalized transition energy according to:

$$\hbar\omega'(k) = \hbar\omega(k) = E + E_i(k) + E_j(k) \quad (903)$$

where $E_i(k)$ and $E_j(k)$ denote the electron and hole subband energies, respectively.

Screened Hartree–Fock Approximation

In the Hartree–Fock approximation, the full Hamiltonian is used. Hartree and Fock proposed an approximation to truncate the infinite hierarchy of the equation of motion [13]. Therefore, the equation of motion for p_k reads:

$$\frac{dp_k}{dt} = -i\omega'_k p_k - i\Omega_k (f^C(k) + f^V(k) - 1) - \gamma p_k \quad (904)$$

where Ω_k denotes the renormalized Rabi frequency.

30: Modeling Quantum Wells

Computing Tabulated Optical Emission Data

The renormalized transition frequency ω' is given by the transition energy:

$$\hbar\omega'(k) = \hbar\omega(k) + \Delta E_{SX}(k) + \Delta E_{CH} \quad (905)$$

which contains the exchange shift term and the Debye shift term. Both terms cause a red shift of the transition energy, which is significant for the screened Hartree–Fock approximation.

Details about the computation of the Coulombic enhancement factor Q_{ij} can be found in the literature [12]. The dephasing rate γ is entered using phenomenological broadening.

Second Born Approximation

The next highest level of approximation is achieved by applying the second Born approximation. The derivation of this formalism can be found in the literature [12][13]. Therefore, the dephasing rate γ is no longer a phenomenological input parameter but is computed within the second Born approximation.

Computing Tabulated Optical Emission Data

The fully coupled solution of the Schrödinger equation, the electronic transport equations, and the optical equations, as described in [The k.p Method on page 825](#), are the most accurate ways of modeling quantum-well laser devices with Sentaurus Device. However, this simulation method can be very time-consuming and, in many cases, re-solving the Schrödinger equation is not necessary because the device is operating under flatband conditions (for example, lasers above threshold).

Nevertheless, to speed up simulations, including full electronic bandstructure information and the most accurate manybody gain calculations, it is possible to load precomputed tables of stimulated and spontaneous emission data into laser and LED simulations (see [Loading Tabulated Stimulated and Spontaneous Emission on page 848](#)).

Computing Emission Tables

Sentaurus Device allows you to compute tables containing the band structure, refractive index change, and stimulated and spontaneous emission spectrum of a single quantum well. The tables are calculated using the framework described in [The k.p Method on page 825](#) and [Optical Emission Including Manybody Effects on page 840](#). The tabulated data is parameterized in four dimensions, that is, the wavelength, electron and hole densities, and lattice temperature. The tables are stored in the HDF5 format.

The syntax for computing emission tables involves:

- The definition of a nonlocal mesh across the quantum well (see [Constructing Nonlocal Mesh on Straight Line on page 831](#)).
- Specification of keywords in the `File`, `Physics`, and `Solve` sections, which enable emission table computation.

An example showing the setup for emission table computation is (the construction of the nonlocal mesh is neglected):

```

File {
    ...
    DephasingRates = "./gamma" # for 2nd Born approximation
    EmissionTable = "table.hdf"
}

...
Physics {
    Temperature=300
    Laser (
        Broadening (
            Gamma = 0.00659995 # [eV]
            Type = Lorentzian # CosHyper, Landsberg, or Lorentzian
        )
        Bandstructure (
            CrystalType = Zincblende # Zincblende or Wurtzite
            Order = kp4x4 # kp4x4, kp6x6, or kp8x8
            NumkValues = 16
            InternalNumkValues = 81 # used for 2nd Born approximation only
        )
        Strain (RefLatticeConst = 5.6544275e-10) # [cm]
        ManyBodyEffects (Type = FCT) # FCT, SHF, or Born
        EmissionTable (
            FrequencySweep (Range=(2.14e15, 2.48e15) Intervals=60 Type=Lin)
            eDensitySweep (Range=(1e+18, 1e+19) Intervals=5 Type=Log)
            hDensitySweep (Range=(1e+18, 1e+19) Intervals=5 Type=Log)
            TemperatureSweep (List (FileName="temp.txt"))
            CalibType = FullTable # FullTable or nEqualp
            NumberOfCB = 1 # used for 2nd Born approximation only
            NumberOfVB = 2 # used for 2nd Born approximation only
        )
        GroupRefract=3.65
    )
}

Solve {
    Bandstructure
    EmissionTable
}

```

30: Modeling Quantum Wells

Computing Tabulated Optical Emission Data

Some comments about this example are:

- In the `File` section ([Table 127 on page 1091](#)), the name of the emission table is specified by the keyword `EmissionTable`. `DephasingRates` is required when the second Born approximation is used. It denotes the directory where the dephasing rates are stored. The HDF files containing the dephasing must follow the naming convention described in [Computing Dephasing Rates on page 847](#).
- In the `Physics-Laser` (or `Physics-LED`) section, the $k \cdot p$ method with various effects is activated by the keywords `Bandstructure`, `Strain`, and `ManyBodyEffects` (see [Syntax for k.p Method on page 831](#)).
- In the `Bandstructure` section ([Table 184 on page 1151](#)), `NumkValues` denotes the number of k -points. `internalNumkValues` must be specified additionally if the second Born approximation is used. Therefore, it is used for the dephasing rate and the optical emission computation.
- In the `ManyBodyEffects` section ([Table 181 on page 1148](#)), three types of manybody effect are available for tabulated emission computation: free carrier theory (FCT), screened Hartree–Fock (SHF), and second Born approximation (Born). The default is the free carrier theory.
- In the `EmissionTable` section ([Table 188 on page 1153](#)), the table parameterization is specified using the keywords `FrequencySweep`, `eDensitySweep`, `hDensitySweep`, and `TemperatureSweep`.

Therefore, `Range=(<float>, <float>)`, `Intervals=<int>`, and `Type=Lin/Log` can be used to specify the parameterization for each dimension. Alternatively, you can define the values for the parameterization in a file as shown for `TemperatureSweep`. Therefore, values must be specified in one column in increasing order.

- `CalibType=nEqaulp` in the `EmissionTable` section ([Table 188 on page 1153](#)) computes stimulated and spontaneous emission for $n=p$ only. This is often sufficient for table calibration. The default is to compute the full table (`CalibType=FullTable`).
- `NumberOfCB` and `NumberOfVB` in the `EmissionTable` section ([Table 188 on page 1153](#)) allow you to specify the number of subbands that are used for the computation of stimulated and spontaneous emissions using the second Born approximation.
- In the `Solve` section ([Table 133 on page 1100](#)), `Bandstructure` and `EmissionTable` are specified to enable the computation of tabulated emission data.

Computing Dephasing Rates

As described in [Optical Emission Including Manybody Effects on page 840](#), the dephasing rate γ is no longer a phenomenological input parameter for optical emission computation if the second Born approximation is used. An example to precompute the dephasing rates is:

```

File {
    ...
    DephasingRates = "./gamma"
}

...
Physics {
    Temperature=300
    Laser (
        Bandstructure(
            CrystalType = Zincblende # Zincblende or Wurtzite
            Order = kp4x4
            NumkValues = 41
            InternalNumkValues = 81
        )
        Strain (RefLatticeConst = 5.6544275e-10)    # [cm]
        ManybodyEffects ( Type=Born )
        EmissionTable (
            FrequencySweep (Range=(2.14e15, 2.48e15) Intervals=60 Type=Lin)
            eDensitySweep (Range=(1e+18, 1e+19) Intervals=5 Type=Log)
            hDensitySweep (Range=(1e+18, 1e+19) Intervals=5 Type=Log)
            TemperatureSweep (List (FileName="temp.txt"))
            CalibType = FullTable # FullTable or nEqualp
        )
        GroupRefractive = 3.6
    )
}

Solve {
    Bandstructure
    DephasingRates
}

```

Some comments about this example are (further comments can be found in [Computing Emission Tables on page 844](#)):

- In the File section ([Table 127 on page 1091](#)), DephasingRates specifies the directory where the dephasing rates are stored. The HDF files containing the dephasing are named `gamma_tX_nY_pZ.hdf`, where `x`, `y`, and `z` denote the corresponding values for temperature, electron density, and hole density, respectively.
- In the ManyBodyEffects section ([Table 182 on page 1150](#)), Type=Born must be specified to use second Born approximation.

30: Modeling Quantum Wells

Loading Tabulated Stimulated and Spontaneous Emission

- In the Bandstructure section ([Table 184 on page 1151](#)), NumkValues=41 is set. Furthermore, internalNumkValues must be specified.
- In the Solve section ([Table 133 on page 1100](#)), Bandstructure and DephasingRates are specified to enable the computation of the dephasing rates.

Loading Tabulated Stimulated and Spontaneous Emission

To speed up simulations, nevertheless, including full electronic bandstructure information and the most accurate manybody gain calculations, it is possible to load precomputed tables of stimulated and spontaneous emission data into laser and LED simulations. The tables can be computed using Sentaurus Device as described in [Computing Tabulated Optical Emission Data on page 844](#).

When the tables have been computed, the loader can be activated by using the SponEmissionCoeff and StimEmissionCoeff keywords in the Physics-Laser section of the command file and by specifying the file names of the tables to be loaded. Different tables can be loaded for different regions or materials. The command file syntax is:

```
Physics{ ...
    Laser( ...
        StimEmissionCoeff (Tabulated)
        SponEmissionCoeff (Tabulated)
    )
    ...
}
File{
    StimEmissionTable = "somename.hdf"# global specification
    SponEmissionTable = "somename.hdf"
}
```

or alternatively:

```
File(Material = "GaAs") {
    StimEmissionTable = "somename.hdf"# material-wise specification
    SponEmissionTable = "somename.hdf"
}
```

or:

```
File(Region = "someregion") {
    StimEmissionTable = "somename.hdf"# region-wise specification
    SponEmissionTable = "somename.hdf"
}
```

Regionwise specifications override materialwise specifications that, in turn, override a global specification.

Importing Gain and Spontaneous Emission Data with PMI

Sentaurus Device can import external stimulated and spontaneous emission data through the physical model interface (PMI). The gain PMI concept is illustrated in [Figure 111](#).

Sentaurus Device calls the user-written gain calculations through the PMI with the variables: electron density n , hole density p , electron temperature eT , hole temperature hT , and transition energy E . The user-written gain calculation then returns the gain g and the derivatives of gain with respect to n , p , eT , and hT to Sentaurus Device. The derivatives are required to ensure proper convergence of the Newton iterations. In this way, the user-import gain is made self-consistent within the laser simulation.

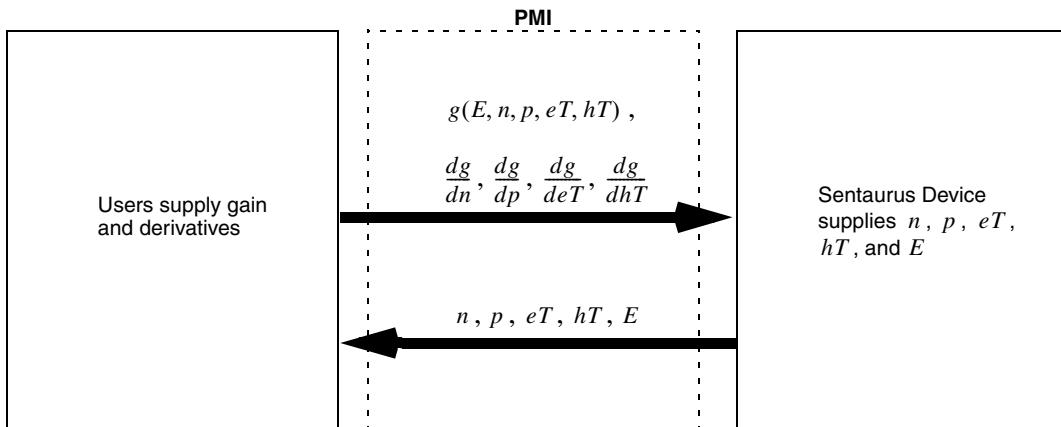


Figure 111 Concept of the gain PMI

Implementing Gain PMI

The PMI uses the object-orientation capability of the C++ language (see [Chapter 34](#) on page 911). A brief outline is given here of the gain PMI.

In the Sentaurus Device header file `PMIModels.h`, the following base class is defined for gain:

```

class PMI_StimEmissionCoeff : public PMI_Vertex_Interface {
public:
    PMI_StimEmissionCoeff (const PMI_Environment& env);
    virtual ~PMI_StimEmissionCoeff ();

    virtual void Compute_rstim
  
```

30: Modeling Quantum Wells

Importing Gain and Spontaneous Emission Data with PMI

```
(double E,
    double n,
    double p,
    double et,
    double ht,
    double& rstim) = 0;

virtual void Compute_drstimdn
(double E,
    double n,
    double p,
    double et,
    double ht,
    double& drstimdn) = 0;

virtual void Compute_drstimdp
(double E,
    double n,
    double p,
    double et,
    double ht,
    double& drstimdp) = 0;

virtual void Compute_drstimdet
(double E,
    double n,
    double p,
    double et,
    double ht,
    double& drstimdet) = 0;

virtual void Compute_drstimdht
(double E,
    double n,
    double p,
    double et,
    double ht,
    double& drstimdht) = 0;
};
```

To implement the PMI model for gain, you must declare a derived class in the user-written header file:

```
#include "PMIModels.h"

class StimEmissionCoeff : public PMI_StimEmissionCoeff {
    // User-defined variables for his/her own routines
    private:
        double a, b, c, d;

    public:
        // Need a constructor and destructor for this class
        StimEmissionCoeff (const PMI_Environment& env);
        ~StimEmissionCoeff ();

        // --- User needs to write the following routines in the .C file ---
        // The value of the function is return as the last pointer argument

        // stimulated emission coeff value
        void Compute_rsttim (double E,
                              double n,
                              double p,
                              double et,
                              double ht,
                              double& rsttim);

        // derivative wrt n
        void Compute_drsttimdn (double E,
                               double n,
                               double p,
                               double et,
                               double ht,
                               double& drsttimdn);

        // derivative wrt p
        void Compute_drsttimdp (double E,
                               double n,
                               double p,
                               double et,
                               double ht,
                               double& drsttimdp);

        // derivative wrt eT
        void Compute_drsttimdet (double E,
                               double n,
                               double p,
                               double et,
                               double ht,
                               double& drsttimdet);
```

```
// derivative wrt hT
void Compute_drstimdht (double E,
                         double n,
                         double p,
                         double et,
                         double ht,
                         double& drstimdht);
}
```

Next, you must write the functions `Compute_rstim`, `Compute_drstimdn`, `Compute_drstimdp`, `Compute_drstimdet`, and `Compute_drstimdht` to return the values of the stimulated emission coefficient and its derivatives to Sentaurus Device using this gain PMI. If you have, for example, a table of gain values, you must implement the above functions to interpolate the values of the gain and derivatives from the table.

The spontaneous emission coefficient can also be imported using the PMI. The implementation is exactly the same as the stimulated emission coefficient, and you only need to replace `StimEmissionCoeff` with `SponEmissionCoeff` in the above code example.

References

- [1] M. Grupen and K. Hess, “Simulation of Carrier Transport and Nonlinearities in Quantum-Well Laser Diodes,” *IEEE Journal of Quantum Electronics*, vol. 34, no. 1, pp. 120–140, 1998.
- [2] L. A. Coldren and S. W. Corzine, *Diode Lasers and Photonic Integrated Circuits*, New York: John Wiley & Sons, 1995.
- [3] M. Ahmed and M. Yamada, “An infinite order perturbation approach to gain calculation in injection semiconductor lasers,” *Journal of Applied Physics*, vol. 84, no. 6, pp. 3004–3015, 1998.
- [4] B. Witzigmann, A. Witzig, and W. Fichtner, “Nonlinear Gain Saturation for 2-Dimensional Laser Simulation,” in *Lasers and Electro-Optics Society (LEOS) 12th Annual Meeting*, vol. 2, San Francisco, CA, USA, pp. 659–660, November 1999.
- [5] Z.-M. Li *et al.*, “Incorporation of Strain Into a Two-Dimensional Model of Quantum-Well Semiconductor Lasers,” *IEEE Journal of Quantum Electronics*, vol. 29, no. 2, pp. 346–354, 1993.
- [6] R. Eppenga, M. F. H. Schuurmans, and S. Colak, “New k.p theory for GaAs/Ga_{1-x}Al_xAs-type quantum wells,” *Physical Review B*, vol. 36, no. 3, pp. 1554–1564, 1987.

- [7] S. L. Chuang and C. S. Chang, "A band-structure model of strained quantum-well wurtzite semiconductors," *Semiconductor Science and Technology*, vol. 12, no. 3, pp. 252–263, 1997.
- [8] A. Bykhovski, B. Gelmont, and M. Shur, "The influence of the strain-induced electric field on the charge distribution in GaN-AlN-GaN structure," *Journal of Applied Physics*, vol. 74, no. 11, pp. 6734–6739, 1993.
- [9] F. Bernardini, V. Fiorentini, and D. Vanderbilt, "Spontaneous polarization and piezoelectric constants of III-V nitrides," *Physical Review B*, vol. 56, no. 16, pp. R10024–R10027, 1997.
- [10] M. Pfeiffer, *Industrial-Strength of Quantum-Well Semiconductor Lasers*, Ph.D. thesis, ETH, Zurich, Switzerland, 2004.
- [11] W. W. Chow and S. W. Koch, *Semiconductor-Laser Fundamentals: Physics of the Gain Materials*, Berlin: Springer, 1999.
- [12] M. Luisier and S. Odermatt, *Simulation of Semiconductor Lasers, Part 1: Many-Body Effects in Semiconductor Lasers*, Diploma thesis, ETH, Zurich, Switzerland, 2003.
- [13] H. Haug and S. W. Koch, *Quantum Theory of the Optical and Electronic Properties of Semiconductors*, Singapore: World Scientific, 3rd ed., 1994.

30: Modeling Quantum Wells

References

Additional Features of Laser or LED Simulations

This chapter describes additional features that are available to aid the simulation of both lasers and light-emitting diodes.

Plotting Gain and Power Spectra

Modal or material gain is an important parameter in laser operations. It affects the threshold current, modulation response, external efficiency, lasing wavelength, and so on. In the simulation, the modal or material gain can be monitored in three ways: as a function of bias, spatial distribution, and a function of energy. Refer to [Simulating Single-Grid Edge-emitting Laser on page 658](#) while proceeding through these gain-plotting options.

Modal Gain as Function of Bias

The modal gain for the lasing frequency is automatically output in the Current file in a laser simulation if the current file has been specified:

```
File {...
    Current = "multiqw_curr"
}
```

At the end of the simulation, the file `multiqw_curr_des.plt` is produced. With Inspect, you can plot `OpticalGain` as a function of bias current, voltage, and so on. This modal gain [cm^{-1}] is the total gain of the device at the lasing frequency. In multimode simulations, a modal gain curve corresponds to each mode, and the modal gain is taken at the respective lasing frequencies of the modes.

Material Gain in Active Region

The peak value of the local material gain at each vertex of the active region can be extracted by including the keyword `MatGain` in the `Plot` section:

```
File {...
    Grid = "mesh_mdr.grd"
    Plot = "multiqw_plot"
```

31: Additional Features of Laser or LED Simulations

Plotting Gain and Power Spectra

```
    }
    ...
    Plot {...  
        MatGain  
    }
    ...
```

At the end of the simulation, the file `multiqw_plot_des.dat` will be produced. This file can be used in Tecplot SV in conjunction with the grid file, `mesh_mdr.grd`, to view the peak material gain at each spatial location of the active region.

Modal Gain as Function of Energy/Wavelength

One important way to look at the gain is to plot the modal gain as a function of the energy. This is possible by including gain-plotting keywords in the `File` and `Solve-quasistationary` sections of the command file:

```
File {...  
    Gain = "gn"  
}  
GainPlot{ Range=(1.22,1.32) Intervals=120 }  
                                # energy range [eV], discretization  
...  
Solve {...  
    Quasistationary {...  
        # ----- Specify plot gain parameters -----  
        PlotGain {Range=(0,1) Intervals=3}  
        Goal {Name="p_Contact" Voltage=1.8})  
        {...}  
    }  
}
```

The activation syntax is similar to that of `OptFarField` (see [Far Field on page 786](#)) and `SaveOptField` (see [Saving and Loading Optical Modes on page 783](#)). The highlights of the syntax are:

- In the `File` statement, modal gain-plotting is activated by the keyword `Gain` and the value assigned to it, "gn" in this case, becomes the base name for the output files of the modal gain as a function of energy.
- In the `Solve-quasistationary` section, the argument `range=(0,1)` in the `PlotGain` keyword is mapped to the initial and final bias conditions. In this example, the initial and final (goal) `p_Contact` voltages are 0 V and 1.8 V, respectively. The number of `intervals=3`, which gives a total of four (= 3+1) modal gain plots at 0 V, 0.6 V, 1.2 V, and 1.8 V. In general, specifying `intervals=n` will produce `(n+1)` plots.

- The GainPlot statement allows you to choose the energy range of the gain curve to plot. In this example, the range has been chosen as 1.22–1.32 eV, and 120 discretization points are to be used.
- The output files in this example are gn_gain_000000_des.plt, ..., gn_gain_000003_des.plt. These files contain the modal gains as a function of energy or wavelength, and can be viewed with Inspect.
- If the full photon-recycling feature is activated in a LED simulation, there will be an additional modified spontaneous emission spectrum in the plot file (see [Full Photon-Recycling Model \(Active Region\) on page 743](#)).

Transient Simulation

Transient simulation is important in the tracking of the time evolution of carrier dynamics and photon output fluctuations in a laser diode. Sentaurus Device has an option to perform the transient simulation of edge-emitting lasers, VCSELs, and LEDs. A small step bias is applied at the input and, through Newton iterations, the time-varying continuity equations and photon rate equations are solved self-consistently with the Poisson equation, QW scattering equations, and Schrödinger equation.

Upon the application of the step bias, the time steps are increased in small intervals to trace the dynamics of the carriers and photons in the transient simulation. At each time step, the full set of variables everywhere in the device can be saved and plotted. In this way, you can calculate which feature of the device is hampering the faster modulation of the device.

Transient simulations performed at different biases yield different dynamics, so you must perform a quasistationary simulation first to reach the required bias current or voltage level, after which the transient simulation is activated. The syntax for transient simulation of general devices is described in [Transient Command on page 79](#). Here, the emphasis is on explaining the syntax that is related to [Simulating Single-Grid Edge-emitting Laser on page 658](#).

Syntax for Laser Transient Simulation

The transient simulation can be activated by the keyword `Transient` in the `Solve` section of the command file:

```
Solve {
    # ----- Get initial guesses, coupled means Newton's iteration -----
    Poisson
    coupled {Hole Electron Poisson}
    coupled {Hole Electron QWhScatter QWeScatter Poisson}
    coupled {Hole Electron QWhScatter QWeScatter Poisson PhotonRate}
```

31: Additional Features of Laser or LED Simulations

Transient Simulation

```
# ----- Ramping the voltage to 1.8 V -----
quasistationary (
    # ----- Specify ratio step size of voltage ramp -----
    InitialStep = 0.001
    MaxStep = 0.05
    Minstep = 1e-5

    # ----- Specify the final voltage ramp goal -----
    Goal {name="p_Contact" voltage=1.8})
{

    # ----- Gummel iterations for self-consistency of Optics -----
    Plugin(BreakOnFailure){
        # --- Newton iterations for coupled equations -----
        Coupled { Electron Hole Poisson QWeScatter QWhScatter
                   PhotonRate }

        Wavelength
        Optics
    }
}

# ----- At 1.8V, perform transient simulation -----
transient (
    # ----- Specify the starting and ending time -----
    InitialTime = 0.0          # [s]
    FinalTime = 2.0e-9         # [s]

    # ----- Control the time step size -----
    InitialStep = 1.0e-3
    MinStep = 1.0e-3
    MaxStep = 1.0e-1

    # ----- Save the plot variables at specified intervals -----
    Plot { range=(0,1) intervals=4 }
)
{
    Plugin(BreakOnFailure){
        Coupled { Electron Hole Poisson QWeScatter QWhScatter
                   PhotonRate }

        # Wavelength
        # Optics
    }
}
}
```

The above syntax has been extracted from [Simulating Single-Grid Edge-emitting Laser on page 658](#):

- A quasistationary simulation is performed first to ramp up the operating voltage to a bias of 1.8 V before the transient simulation is activated.
- In the transient section, you must specify the start and end time. The response of the carriers and photons subjected to a small step bias input will generally settle to a steady state in the time frame of a few thousand picoseconds. This settling time depends on bias conditions and the type of laser diode. You are encouraged to use different values of FinalTime so that the important parts of the transient response are observed.
- In the transient section, you must specify the time-step size as well in the same format as is used to specify the bias step size in the quasistationary simulation. In this case, the time-step size is the fraction indicated in InitialStep, MinStep, and MaxStep multiplied by (FinalTime – InitialTime), which gives 2, 2, and 200 ps, respectively. The scattering time for the carriers is in the order of 1×10^{-13} s, so a time step shorter than this time will not be meaningful. Of course, if the time step is too big, it cannot resolve the finer features of the transient response.
- In the transient section, the Plot statement enables you to save the variables that have been defined in the Plot section at required intervals of the transient simulation. In this example, intervals=4 produces five plot files at the time intervals of 0, 400, 800, 1200, 1600, and 2000 ps.
- The transient response computed is saved in the current file. The major results of laser output are saved as a function of time in this current file. You can then perform a FFT of the time response to obtain the modulation response of the laser diode.

Performing Temperature Simulation

There are a few different types of temperatures, but Sentaurus Device only treats two types:

- Lattice temperature, which describes the vibrational states of the crystal lattice
- Carrier temperatures, which determine the distribution of the excited carriers

The lattice temperature is solved by the lattice temperature model described in [Uniform Self-Heating on page 182](#). The carrier temperatures are solved by the hydrodynamic equations discussed in [Hydrodynamic Transport Model on page 184](#).

NOTE For stability reasons, the QW scattering model (see [Quantum-Well Scattering Model on page 806](#)) must be activated when the lattice temperature and hydrodynamic models are solved in a laser simulation.

31: Additional Features of Laser or LED Simulations

Performing Temperature Simulation

Lattice Temperature Simulation

To solve the lattice temperature model, the following three steps must be taken:

1. A `thermode` must be defined in the grid file and command file.
2. In the `Physics` section, the keyword `Thermodynamic` is optional. If it is included, a lattice temperature gradient term is appended to the carrier current flux density.
3. In the `Solve` section, the keyword `Temperature` must be added to the `Coupled` statement.

These steps are embedded in the following syntax:

```
# --- Need to specify where to impose Dirichlet boundary condition for
# temperature solution ---
Thermode {
    {Name="top_thermode" AreaFactor=200 Temperature=300 SurfaceResistance =
0.14}
    {Name="bot_thermode" AreaFactor=200 Temperature=300 SurfaceResistance =
0.09}
}

Plot {
    # ----- Temperature variables -----
    Temperature
}
...
Physics {...}
    Laser ...
    Optics ...
)
# ----- Turn on thermodynamic simulation -----
    Thermodynamic
    RecGenHeat
}
...
Solve {
    # ----- Get initial guesses, coupled means Newton's iteration -----
    Poisson
    coupled {Hole Electron QWhScatter QWeScatter Poisson}
    coupled {Hole Electron QWhScatter QWeScatter Poisson PhotonRate}
    coupled {Hole Electron QWhScatter QWeScatter Poisson PhotonRate Temperature}

    # ----- Ramping the voltage -----
    quasistationary (
        # ----- Specify ratio step size of voltage ramp -----
        InitialStep = 0.001
```

```

MaxStep = 0.05
Minstep = 1e-5

# ----- Specify the final voltage ramp goal -----
Goal {name="p_Contact" voltage=1.8}
{
    # ----- Gummel iterations for self-consistency of Optics -----
    Plugin(BreakOnFailure){
        # --- Newton iterations for coupled equations -----
        Coupled {Electron Hole Poisson QWeScatter QWhScatter
                  PhotonRate Temperature}
        Wavelength
        Optics
    }
}
}

```

Refer to [Simulating Single-Grid Edge-emitting Laser on page 658](#) for the rest of the laser simulation syntaxes, which have been omitted here. Some comments about the above syntax are:

- A thermode is a boundary where the Dirichlet boundary condition is set for the lattice temperature equation. At all other boundaries without a thermode, Neumann boundary condition is assumed. It can be set in the same way as the contact electrodes are set in Sentaurus Structure Editor, and this is shown in [Figure 112](#). The SurfaceResistance [$\text{cm}^2 \text{K/W}$] can be used to tune the lattice temperature profile in the device.
- In the Physics section, the keyword Thermodynamic contributes an additional term (gradient of the lattice temperature) to the electron and hole flux densities. The keyword RecGenHeat adds heat produced or absorbed by generation–recombination to the lattice temperature equation.
- In the Solve section, the keyword Temperature in the Coupled statement adds the lattice temperature equation to the Newton system to be solved iteratively.

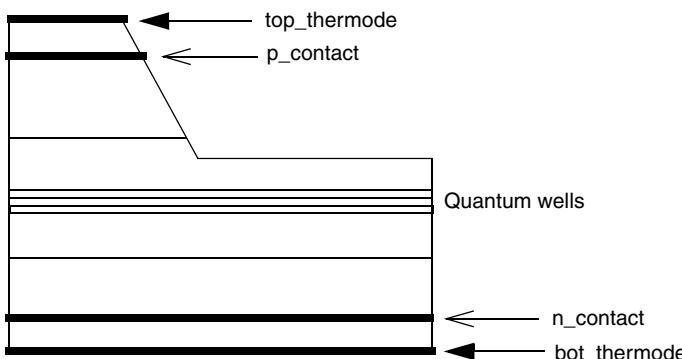


Figure 112 Setting thermodes in the same way as setting contacts; labeled top_thermode and bot_thermode, respectively

31: Additional Features of Laser or LED Simulations

Performing Temperature Simulation

Carrier Temperature Simulation

The carrier temperatures can be solved by the hydrodynamic model, and the activation syntaxes are located in similar locations as the lattice temperature model:

```
Plot {
    # ----- Temperature variables -----
    eTemperature
    hTemperature
}
...
Physics {...}
    Laser (...)

        Optics (...)

    )
    # ----- Specify ambient device temperature -----
    Temperature = 298
    # ----- Turn on hydrodynamic simulation -----
    Hydrodynamic
        RecGenHeat
}
...
Solve {
    # ----- Get initial guesses, coupled means Newton's iteration -----
    Poisson
    coupled {Hole Electron QWhScatter QWeScatter Poisson}
    coupled {Hole Electron QWhScatter QWeScatter Poisson PhotonRate}
    coupled {Hole Electron QWhScatter QWeScatter Poisson PhotonRate
            eTemperature hTemperature}

    # ----- Ramping the voltage -----
    quasistationary (
        # ----- Specify ratio step size of voltage ramp -----
        InitialStep = 0.001
        MaxStep = 0.05
        Minstep = 1e-5

        # ----- Specify the final voltage ramp goal -----
        Goal {name="p_Contact" voltage=1.8})
{
```

```

# ----- Gummel iterations for self-consistency of Optics -----
Plugin(BreakOnFailure){
    # --- Newton iterations for coupled equations ---
    Coupled {Electron Hole Poisson QWeScatter QWhScatter
              PhotonRate eTemperature hTemperature}
    Wavelength
    Optics
}
}
}

```

Some comments about the above syntax are:

- You do not need to specify any thermode if the hydrodynamic model is solved without the lattice temperature equation. However, it is possible to solve both the lattice temperature and hydrodynamic models together. You must include the keywords for both models.
- Note that the carrier temperatures are solved everywhere in the device except the quantum wells. The physics of transport into the quantum wells cannot be handled by the hydrodynamic model.
- In the Plot section, you can choose to save the electron and hole temperatures everywhere in the device with the keywords eTemperature and hTemperature.
- In the Physics section, if the lattice temperature is not solved, you can specify an ambient device lattice temperature with the keyword Temperature=<float>. The hydrodynamic model is activated by the keyword Hydrodynamic in this section. The keyword RecGenHeat adds the heat produced or absorbed as a result of carrier generation-recombination to the hydrodynamic equations.
- In the Solve section, the hydrodynamic model requires two keywords, eTemperature and hTemperature, to be included in the Coupled statement.

Switching from Voltage to Current Ramping

Before the lasing threshold, the voltage varies almost linearly with the spontaneous output power. At and beyond the lasing threshold, a small increase in voltage will lead to an exponential-order increase in the laser output power. Therefore, it is desirable to switch between voltage and current bias ramping in a laser simulation. This is achieved using the following syntax in the Solve section of the command file:

```

Solve {
    # --- Solve for initial guesses of the coupled system ---
    Poisson
    Coupled {Poisson Hole Electron}
    Coupled {Poisson Hole Electron QWeScatter QWhScatter}
    Coupled {Poisson Hole Electron QWeScatter QWhScatter PhotonRate}

```

31: Additional Features of Laser or LED Simulations

Switching from Voltage to Current Ramping

```
# --- Ramp using voltage bias to near lasing threshold ---
Quasistationary (
    InitialStep = 0.01
    MaxStep = 0.1
    Minstep = 1e-9
    Goal {name="p_contact" voltage=1.3})
{
    Plugin(BreakOnFailure){
        Coupled { Poisson Hole Electron QWeScatter QWhScatter
                  PhotonRate }
        Optics
        Wavelength
    }
}

# --- Change the p_contact from voltage to current type ---
Set ("p_contact" mode current)

# --- Then ramp using current bias ---
Quasistationary (
    InitialStep = 0.01
    MaxStep = 0.05
    Minstep = 1e-8
    Goal {name="p_contact" current=2.5e-4})
{
    Plugin(BreakOnFailure){
        Coupled { Poisson Hole Electron QWeScatter QWhScatter
                  PhotonRate }
        Optics
        Wavelength
    }
}
}
```

The threshold voltage of a laser diode can be estimated approximately by considering the laser diode as a p-i-n diode with the quantum well as a strong recombination center. In order for diffusion current to flow into the quantum well, the intrinsic Fermi levels of the bulk region near the p or n contact must approximately align with that of the quantum well. This means a voltage that is approximately the difference in these Fermi levels must be applied for the device to start the current flow towards the quantum wells to fill them.

Of course, the lasing will not start until the quantum wells are filled to a level such that population inversion is achieved and the threshold losses are overcome. Resistance of the semiconductor layers will also contribute to the voltage. Nevertheless, this consideration will give a rough estimate of the threshold voltage for current flow in the laser diode.

Robust Wavelength Search Algorithm

In cases where the default wavelength search algorithm fails in a manybody laser simulation or due to a strong strain shift of the quantum-well subbands, the keyword `NewWavelengthSearch(InitialWavelength=<float>)` can be activated in the `Math` statement. This switches on a more robust algorithm that, in most cases, resolves the convergence problem. The specification of an initial wavelength for the new search algorithm is optional. If specified, it must be given in units of nanometers and its value must be larger than the expected actual transition wavelength.

Reducing Number of Result Files

Sentaurus Device can be used to reduce the number of output files for laser simulations. Plots are represented by states that are collected in one output file. This feature is supported for VCSEL near field, laser far field, and gain and spectrum plots using the TDR file format. It is activated by adding the option (`collected`) to the keyword that specifies the output file.

An example of the syntax is:

```
File {
    OptFarField (collected) = "farfield"
}
...
Solve {
    ...
    Quasistationary (
        PlotFarField {Range=(0, 1) Intervals=3}
    )
    ...
}
```

The output file in this example is called `farfield_ff_coll_des.plt`. It contains four states and can be visualized by Tecplot SV.

Scripts

There is a list of scripts that are written in Tcl and can be used to extract various useful parameters from laser and LED simulations, as well as to control different tools in the Synopsys software suite. The laser-related or LED-related scripts available from the Synopsys Technical Support Center are:

- Automatic extraction of the threshold current
- Calculation of the slope efficiency of the L–I curve
- Automatic addition of DBR layers in the VCSEL structure
- Automatic addition of PML to the laser structure
- Generation of multiple–quantum well (MQW) edge-emitting lasers
- Generation of MQW VCSEL structures
- Ternary-material and quaternary-material parameter calculations
- Extraction of far-field angle from far-field patterns

Part IV Mesh and Numeric Methods

This part of the Sentaurus Device manual contains the following chapters:

[Chapter 32 Automatic Grid Generation and Adaptation Module AGM on page 869](#)

[Chapter 33 Numeric Methods on page 885](#)

Automatic Grid Generation and Adaptation Module AGM

This chapter describes the automatic generation and adaptation of two-dimensional quadtree-based simulation grids for physical devices.

The approach is based on a local anisotropic grid adaptation technique for the drift-diffusion model [1][2][3] and has been formally extended to thermodynamic and hydrodynamic simulations.

Overview

NOTE In its current status, AGM is not designed to improve the speed of simulations but rather to control the accuracy of the solution. In fact, using AGM slows down the simulation time considerably as the control of the grid sizes is difficult and the recomputation of solutions on adaptively generated grids is, in the presence of strong nonlinearities, a time-consuming task. Therefore, it is not recommended to use AGM throughout large simulation projects in a fully automatic adaptation modus. AGM in its current status may be used in a semi-automatic fashion to understand which parts of a device grid require more refinement to improve solution accuracies and to construct for a set of simulations appropriately fixed simulation grids. Its integration in Sentaurus Device is incomplete as incompatibilities occur with certain features of Sentaurus Device.

The accuracy of approximate solutions computed by many simulation tools depends strongly on the simulation grid used in the discretization of the underlying problem. The major aim of grid adaptation is to obtain numeric solutions with a controlled accuracy tolerance using a minimal amount of computer resources using *a posteriori* error indicators to construct appropriate simulation grids. The main building blocks of a local grid adaptation module for stationary problems are the adaptation criteria (local error indicators that somehow determine the quality of grid elements), the adaptation scheme (determining if and how the grid will be modified on the basis of the adaptation criteria), and the recomputation procedure of the approximate solution on adaptively generated grids. In the framework of finite-element methods for linear, scalar elliptic boundary value problems grid adaptation has reached a mature status [4].

The semiconductor device problem consists of a nonlinearly coupled system of partial differential equations and the true solution of the problem exhibits layer behavior and singularities, posing additional difficulties for grid adaptation modules. Several adaptation criteria have been proposed in the literature [1]. For the overall robustness of adaptive simulations, the recomputation of the solution is a very serious (and, sometimes, very time-consuming) problem.

The adaptation procedure used in Sentaurus Device is based on the approach developed in [1], [2], and [3]. It uses the idea of equidistributing local dissipation rate errors and aims at accurate computations of the terminal currents of the device. The quadtree mesh structure of Mesh is used to enable anisotropic grid adaptation on boundary Delaunay meshes required by the discretization used in Sentaurus Device. The recomputation procedure relies on local and global characterizations of dominating nonlinearities and includes relaxation techniques based on the solution of local boundary value problems and a global homotopy technique for large avalanche generation.

Adaptation Procedure

The adaptation flow is sketched in the following pseudo-code:

```

compute solution on actual grid(s)

coupled adaptation loop{
    // adaptation decision
    for all adaptive devices {
        compute adaptation criteria (including h/2-grid computations)
        check if adaptation is required
    }
    check if coupled adaptation is required

    // adaptation strategy
    for all adaptive devices {
        // grid adaptation
        criteria adaptation (tree loop)
        (directed) neighbor size refinement
        new simulation grid (conforming, delaunization)

        // device level data smoothing
        data interpolation (initial guess)
        electrostatic potential correction (EPC)
        nonlinear node block Jacobi smoothing (NBJI)
        avalanche homotopy
    }
}

```

```
// system level smoothing
Newton iteration to achieve self consistent solution of the fully coupled
system
}
```

The coupled adaptation loop is iterated until the system fulfills the adaptation criteria or the maximal number of iterations is reached (`MaxCLoops`).

Adaptation Decision

The decision as to whether the grid of a device must be updated depends on the (global) relative error of the observables F specified for the adaptation criteria. Refinement and coarsening adaptation are distinguished depending on the size of the relative error, that is, the grid will be refined if for one global error estimate $\eta(F)$ of the (Dirichlet or residual) criteria:

$$\eta(F) > \varepsilon_R(\varepsilon_A + |F^h|) \quad (906)$$

holds, where ε_R and ε_A are the relative and absolute error (specified by `RelError` and `AbsError`), respectively, and F^h is the actual value of the integral quantity (h is the mesh size parameter).

A coarsening adaptation occurs if all criteria satisfy:

$$\eta(F) \leq \frac{1}{20}\varepsilon_R(\varepsilon_A + |F^h|) \quad (907)$$

Adaptation Strategy

If the grid must be adapted, first, the adaptation criteria are applied to the elements of the grid tree performing one or several levels of anisotropic refinement and/or coarsening (tree adaptation loop or RCLoop). To make the mesh regular, a directed (anisotropic) neighbor size refinement is performed and a final delaunization step generates the new simulation mesh. Even in refinement adaptation, coarsening is allowed (if not disabled by you) for elements with small errors. In a coarsening adaptation, no refinement is performed.

Adaptation Criteria

The adaptation criteria determine if and how the grid will be modified. In [1], adaptation criteria for the nonlinearly coupled system of equations for the drift-diffusion model have been proposed, aiming at accurate computations of the terminal currents of the device. They use the close relationship between the system dissipation rate and the terminal currents, and estimate the error of the dissipation rate. This is performed by either solving related local Dirichlet problems or using the residual error estimation technique. Both techniques are well known in the framework of finite-element discretizations for scalar elliptic boundary value problems [4]. The following adaptation criteria are supported by Sentaurus Device.

Criteria Based on Local Dirichlet Problems

These criteria construct locally refined meshes and solve local Dirichlet problems to estimate the error of the interesting integral quantity (criterion type `Dirichlet`). Their computation is quite expensive as they require the solution of additional problems, that is, they are based on implicit error indicators.

The Dirichlet adaptation criteria estimate the *error of the quantity F* per grid element T as:

$$\eta_T(F) = \int_T |(\omega_F^h - \omega_F^{h/2})| dx \quad (908)$$

where ω_F^h and $\omega_F^{h/2}$ are the quantity density on the actual simulation grid and the locally refined $h/2$ -grid, respectively. A weaker form of the error is given by the expression:

$$\eta_T^D(F) = \int_T (\omega_F^h - \omega_F^{h/2})^2 dx \quad (909)$$

and is called *deviation of the quantity F* on element T . Dirichlet-type adaptation criteria are implemented for the so-called AGM dissipation rate, which is a weighted form of the physical system dissipation rate and the domain integral current (of a given contact). The AGM dissipation rate (`AGMDissipationRate`) is given as:

$$D_{\text{AGM}} = \hat{w}_n \int_{\Omega} \mu_n n |\vec{J}_n|^2 dx + \hat{w}_p \int_{\Omega} \mu_p p |\vec{J}_p|^2 dx + \hat{w}_r kT \int_{\Omega} R_{\text{abs}} \left| \ln \left(\frac{np}{n_{i,\text{eff}} p_{i,\text{eff}}} \right) \right| dx \quad (910)$$

where $R_{\text{abs}} = \sum \hat{w}_{R_i} |R_i|$ is the weighted absolute sum of individual generation–recombination processes with their individual weights \hat{w}_{R_i} . You can modify all weights \hat{w}_i .

The (electron) domain integral current at contact C is given as:

$$I_n^C = - \int_{\Omega} R h_n^C + \vec{J}_n \cdot \nabla h_n^C dx \quad (911)$$

where h_n^C is a suitable test function with $h_n^C(x) = 1$ for locations x on contact C , $h_n^C(x) = 0$ for all locations x on contacts $C' \neq C$. The sum of electron and corresponding hole domain integral currents give the (total) domain integral current (`DomainIntegralCurrent`) at the specified contact.

Residual Adaptation Criteria

The residual adaptation criteria (so far, only available for the `AGMDissipationRate`) are explicit error estimators as they refer only to the actual solution (criterion type `Residual`). In analogy to standard methods, they measure jumps of the density of interest across inter-element boundaries. The error for element T is estimated as:

$$\eta_T(F) = \frac{|T|}{|N_e(T)|} \sum_{T' \in N_e(T)} |\omega_F^h(T') - \omega_F^h(T)| \quad (912)$$

where $\omega_F^h(T)$ denotes the approximate element functional density (on element T and T' , respectively), $N_e(T)$ is the set of (semiconductor) elements sharing an edge with T , $|N_e(T)|$ is the number of elements, and $|T|$ is the volume of T .

Criteria Based on Element Variation

Extending the adaptation criteria of [1], an adaptation criterion based on the variation of arbitrary (on vertices defined) functionals has been implemented (criterion type `Element`). The criterion uses simply the differences of vertex-based data f as an error indicator, that is:

$$\eta_T(f) = \max \{ |f(x_i) - f(x_j)| : \overline{x_i x_j} \text{ edge of } T \} \quad (913)$$

Elements are refined if the value exceeds a user-specified value (using `MaxTransDiff`).

Solution Recomputation

For the recomputation of the solution, the data is interpolated onto the new simulation, and iterative smoothing techniques are applied to improve the robustness of the procedure.

Device-Level Data Smoothing

In the first step, the electrostatic potential is adjusted to interpolated data by applying the so-called electrostatic potential correction (EPC), that is, a mixed linear and nonlinear Poisson equation is solved using a Newton algorithm. To achieve almost self-consistent solutions for the coupled equations of the device, local Dirichlet problems are solved approximately, resulting in the so-called nonlinear NBJI (node block Jacobi iteration). The node block iterations are performed only on a subset of all grid vertices.

For remarkable avalanche generation, a homotopy technique (or continuation technique), here called avalanche homotopy, is applied to improve the robustness of the recomputation procedure, that is, the true avalanche generation is globally decoupled from the equations and is integrated stepwise into the solution process. As the avalanche generation F_{ava} is an additive term, the equations to be solved $F(x) = F_b(x) + F_{\text{ava}}(x) = 0$ can be split into a basic part F_b and the avalanche generation term F_{ava} , where x represents the unknown solution variables. With the fixed avalanche generation \widehat{F}_{ava} interpolated from the old grid, the avalanche homotopy now reads:

$$H_{\text{ava}}(x, t) = F_b(x) + (1 - t)\widehat{F}_{\text{ava}} + tF_{\text{ava}}(x) \quad (914)$$

where $t \in [0;1]$ is the homotopy parameter ramped from 0 to 1. For $t = 0$, the homotopy is reduced to a simplified problem, while for $t = 1$ the fully coupled system is solved. Thus, the avalanche homotopy is similar to a quasistationary simulation where the avalanche generation is ramped (instead of specified parameters).

System-Level Data Smoothing

On the system level, a self-consistent solution for the original fully coupled system of equations is computed by the Newton algorithm.

Adaptive Device Instances

A device instance is adaptive if the keyword `GridAdaptation` is specified as a section of the instance description. Several parameters can be passed to the instance by specifying parameter or body entries for the keyword, that is:

```
GridAdaptation ( <agm-device-par-list> ) { <agm-device-body-list> }
```

For adaptive devices, a mesh description from the `Mesh` command and boundary files is necessary.

AGM Device Parameters

The possible parameter entries modify device-specific quantities.

Parameters Affecting Grid Generation

The specification `MaxNumberMacroElements=<int>` limits the number of leaf elements in the quadtree (before possible neighbor size refinement and delaunization), that is, no refinement adaptation is performed if the specified value is exceeded.

`MaxCLoops` specifies the maximal number of adaptations of the instance in coupled adaptation loops.

`Weights` modifies the AGM dissipation rate (see [Eq. 910, p. 872](#)) of the instance used in the adaptation criteria. An example is:

```
Weights (eCurrent = 1. hCurrent = 1.e2 Recombination = 1.e3 Avalanche = 1.e-3)
```

where `eCurrent`, `hCurrent`, and `Recombination` refer to the weights \hat{w}_n , \hat{w}_p , and \hat{w}_r , respectively; while `Avalanche`, for example, refers to the corresponding weight of the avalanche generation in R_{abs} . By default, all weights are 1.

The neighbor size refinement can be switched on and off by `[-] NeighborSizeRefinement`, and the material group-dependent ratios can be modified explicitly by using `NeighborSizeRatio`. The material group is `Semiconductor`, or `Insulator`, or `Conductor`.

For the Dirichlet adaptation criteria requiring solutions on locally refined meshes, two modes are available, namely, the element and the global patch mode. The element patch mode computes for each element a solution of local $h/2$ -grid Dirichlet problems; the global patch

32: Automatic Grid Generation and Adaptation Module AGM

Adaptive Device Instances

mode computes the global solution on the $h/2$ -grid and uses this in the adaptation criteria as a reference solution.

In the RCLoop, the tree adaptation loop is specified. The number of iterations applied to the element tree can be specified by using Iterations = <int> (default is 2), and by using [-] Coarsening if coarsening is allowed in refinement adaptations (default is enabled).

Table 113 AGM device parameter entries

Keyword syntax	Description	Default
MaxNumberMacroElements = <int>	Maximal number of leaf elements in macro element tree.	100000
MaxCLoops = <int>	Maximal number of iterations per adaptive coupled system.	100000
Weights (<weight-list>)	Weights in AGM dissipation rate.	1.
[-] NeighborSizeRefinement	Enables/disables neighbor size refinement.	enabled
NeighborSizeRatio(<matgroup>) = <real>	Allowed anisotropic edge length ratios for neighboring elements.	3. for semiconductor, 10^6 else
PatchMode = Element Global	Affects only Dirichlet adaptation criteria: solve mode on $h/2$ -grid for reference solutions of (local/global) Dirichlet problems.	Element
RCLoop {<rcloop-entries>}	Specification of tree loop properties.	-
[-] Poisson	Enables/disables EPC smoothing step.	enabled
[-] Smooth	Enables/disables NBJI smoothing step.	enabled

Device Parameters Affecting Smoothing

The flags [-] Poisson and [-] Smooth enable or disable the EPC smoothing step and the NBJI, respectively.

The avalanche homotopy can be influenced only by a temporary interface via environment variables. The environment variable DES_AGM_AVAHOMOTOPY_NB_IT modifies the number of iterations used within Newton iterations of the homotopy, for example, for a C-shell, the command:

```
setenv DES_AGM_AVAHOMOTOPY_NB_IT 10
```

sets the number of iterations (which is internally multiplied by 5). Setting DES_AGM_AVAHOMOTOPY_NB_IT to 0 disables the avalanche homotopy (default is 5). The environment variable DES_AGM_AVAHOMOTOPY_LINPAR causes a linear parameter ramping in the homotopy, and DES_AGM_AVAHOMOTOPY_EXTRAPOL switches the extrapolation on within the homotopy.

Grid Specification

Adaptive device instances require a grid description in the form of Mesh command and boundary files. This is specified by the keyword `Boundary` in the `File` section of the instance. If `Boundary` is specified, the keywords `Grid` and `Doping` of the `File` section have a different interpretation: These strings now serve as the basis for output file names of grid and doping information (required to visualize corresponding plot files).

An example `File` section is:

```
File {
    Boundary = "mos_agm_msh"      # INPUT files "mos_agm_msh.{cmd,bnd}"
    Grid     = "test_agm_des"     # reinterpreted as OUTPUT !
    Doping   = "test_agm_des"     # reinterpreted as OUTPUT !
    ...
    GridCompressed               # writing compressed grid and doping files
}
```

With the keyword `GridCompressed`, the grid and doping files are written as compressed files (`.gz`).

The specification in the Mesh command file is used to construct the initial mesh for the device. During adaptation, the maximal edge length specifications are respected, that is, in this way it is possible to control the coarsest possible grid. The minimal edge length specifications describe the finest possible mesh during adaptation, that is, elements are not further refined if half of the edge length is smaller than the specified minimal value. The doping-specific refinement information is completely ignored during adaptation, that is, it is only relevant for initial grid generation.

Sentaurus Device supports grid adaptation for the quadtree grid generation approach of Mesh, though the grid generation algorithm used in Sentaurus Device differs slightly from the algorithm of the (stand-alone) mesh generator Mesh:

- Minimal edge length specifications of refinement definitions are interpreted individually per referencing refinement placement (instead of using the smallest, local, minimal edge length value of all refinement definitions).
- The local lower bound of edge lengths allowed in AGM is the smallest, local, minimal edge length of all referenced refinement definitions.
- The `NOFFSET` section in the command file causes an error during parsing and should be removed.

NOTE Sentaurus Device supports the Mesh command file syntax of Mesh Release 7.0.

Adaptation Criteria

The adaptation criteria are specified as AGM device body entries and determine if adaptation (refinement and/or coarsening) is required, for example:

```
GridAdaptation (...) {
    Criteria {           # list of adaptation criteria
        Dirichlet (DataName = "DomainIntegralCurrent" Contact = "drain"
                    AbsError = 1.e-7 RelError = 0.2)
        Residual (DataName = "AGMDissipationRate"
                    AbsError = 1.e-20 RelError = 0.4)
        Element (DataName = "ElectrostaticPotential" MaxTransDiff = 0.1)
    }
}
```

General

Each adaptation criterion refers to a physical quantity specified by DataName = "<quantity-name>". The supported data for the different types of criterion are listed in [Table 114](#).

[Table 114](#) Supported data for different adaptation types of criterion

Criterion type	Supported data
Dirichlet	AGMDissipationRate, DomainIntegralCurrent
Residual	AGMDissipationRate
Element	Any vertex-based scalar datasets known in Sentaurus Device.

The relative and absolute error tolerances ε_R and ε_A are specified using RelError=<real> or AbsError=<real> (not used for element criteria). Each adaptation criterion plots specific data to plot files in addition to the specified data of the Plot section.

Dirichlet

For the functional DomainIntegralCurrent, an additional contact name must be supplied to the criterion using the keyword Contact. The flag CurrentWeighting must be switched on in the Math section of the device. Dirichlet-type criteria plot the specified data, and its error and deviation.

NOTE With the current implementation of the element patch mode, Dirichlet error estimation is very time-consuming and, therefore, such error indicators should be used only if they improve the grid adaptation compared to other criteria.

Residual

No additional options are available. Residual-type criteria plot the specified data and its error.

Element

The element is refined if $\eta_T(f) > d_{\text{mtd}}$ where d_{mtd} is the value specified by `MaxTransDiff = <real>`, for example:

```
Criteria {
    Element (DataName = "ElectrostaticPotential" # vertex based data
              MaxTransDiff = 0.1                      # unit is taken from DATEX
    )
}
```

All data that can be plotted is available as a vertex-based dataset and can be used in the element-type criterion. The specified data is plotted by this criterion type.

Adaptive Solve Statements

Grid adaptation is only performed for explicitly adaptive solve statements using the keyword `GridAdaptation`. Only the highest level `Coupled` and `Quasistationary` solve statements can be adaptive.

General Adaptive Solve Statements

The following parameter entries are interpreted by all adaptive solve statements.

With `MaxCLoops`, the maximal number of adaptation iterations for each adaptive coupled system is given (default is 100000). The flag `[-] Plot` enables or disables device plots for all intermediate device grids (default is disabled). The flag `[-] CurrentPlot` allows for the plotting of current file data on intermediate grids to the current file (default is disabled).

Adaptive Coupled Solve Statements

A Coupled solve statement can be adaptive if (a) it is the highest level solve statement and (b) it contains either none or at least the three drift-diffusion equations (Poisson, Electron, and Hole) for all adaptive devices:

```
Coupled ( ... GridAdaptation ( MaxCLOops = 5 ) )
{ Poisson Electron Hole }
```

Adaptive Quasistationary Solve Statements

In the current implementation, a Quasistationary can be adaptive only if (a) it is the highest level solve statement and (b) its system consists of a Coupled solve statement (containing none or, at least, the Poisson, Electron, and Hole equations of adaptive devices), that is, Plugin statements are not yet supported.

In adaptive quasistationary simulations, it may be useful to restrict the adaptation to certain ranges of the parameter values. This can be achieved by specifying parameter ranges or iteration numbers in a similar fashion as in Plot in solve statements using the Time, IterationStep, and Iterations keywords, for example:

```
Quasistationary (
    GridAdaptation (
        IterationStep = 10
        Iterations = ( 2 ; 7 )
        Time = ( Range = ( 0.2 0.4 ) ; range = (0. 1. )
                  intervals = 5 ; 0.1 ; 0.99 )
                  ...
    ) ...
) { ... }
```

The fixed times specified by Time do not force adaptation at these values (which can be achieved by other methods, for example, adding plot statements for the desired parameter values), while the free time ranges (specified by Range without the keyword Intervals) force adaptation if the actual parameter falls into the specified open interval.

Limitations and Recommendations

Limitations

The grid adaptation approach implemented in Sentaurus Device has been developed for the drift-diffusion model and 2D quadtree-based simulation grids, and is still under development. For convenience, the approach has been formally extended to support other transport models and features available in Sentaurus Device, that is, AGM is formally compatible with the drift-diffusion, thermodynamic, and hydrodynamic transport models. Nevertheless, it must be noted that AGM has been applied so far only for the drift-diffusion model and silicon devices. Total incompatibility is to be expected with the Schrödinger equation solver, heterostructures, interface conditions, and laser equations. These incompatibilities are only partially checked after command file parsing.

Recommendations

Accuracy of Terminal Currents as Adaptation Goal

The choice of appropriate adaptation criteria depends on the adaptation goal. In most adaptive simulation procedures, the local discretization errors are used as adaptation criteria. This approach is not practicable for device simulation as the criteria lead to overwhelmingly large grid sizes. The residual and Dirichlet adaptation criteria for the functionals AGMDissipationRate and DomainIntegralCurrent aim for accurate computations of the device terminal currents, a minimal requirement for all simulation cases, allowing in principle some unresolved solution layers, which do not contribute to the terminal current computation.

AGM Simulation Times

Each coupled adaptation iteration requires remarkable simulation time. In contrast to linear or easy-to-solve problems, for device simulation, most time is consumed in the recomputation procedure of the solution due to the extreme nonlinearities of the problem and not in the pure adaptation of the grid. The Dirichlet adaptation criteria require the solution of local or global nonlinear problems that also consume large parts of the simulation time even if the grid is not to be updated. Before using very time-consuming adaptation criteria, it is recommended to use first explicit adaptation criteria (not requiring the solution of additional equations, such as residual or element variation criteria) to see which kinds of mesh are created and if the AGM module runs robustly on the given simulation.

32: Automatic Grid Generation and Adaptation Module AGM

Limitations and Recommendations

As a second step, it may be useful to add Dirichlet adaptation criteria as they provide more accurate (mathematical) error bounds than the other error indicators.

The most time-consuming parts in many AGM simulations are (in order of importance):

1. Dirichlet adaptation criteria with element path mode computations (very expensive).
2. Avalanche homotopy if the computation for $t = 1$ fails to converge (can be very expensive).
3. Dirichlet adaptation criteria with global patch mode computations.
4. NBJI smoothing step (for large grid sizes).
5. EPC smoothing step.
6. Grid generation.

Dirichlet Versus Residual Adaptation Criteria

The Dirichlet adaptation criteria are implicit adaptation criteria, that is, they require solutions of local boundary value problems, while the residual adaptation criteria are explicit, that is, refer only to the solution on the actual grid. Therefore, the Dirichlet criteria are much more expensive.

Experimentally, the Dirichlet and residual criteria for `AGMDissipationRate` are comparable in wide regions of the device, that is:

$$\eta_F^{\text{Residual}}(D_{\text{AGM}}) \approx c \eta_F^{\text{Dirichlet}}(D_{\text{AGM}}) \quad (915)$$

where $c \approx 4$ for (essentially) 1D and $c \approx 2$ for 2D simulations. This relationship makes the residual error indicator a very favorable choice as it is much cheaper and, in general, smoother than the Dirichlet error indicator.

Large Grid Sizes

The low convergence order of the discretization causes quite large grid sizes even for low accuracy requirements. Especially in the vicinity of solution layers and singularities (at boundary points with changing boundary conditions), the point density is hard to control. You have several possibilities to influence the sizes of the resulting grids:

- Increase minimal edge length specification in the Mesh command file: Elements that reach the allowed minimal edge length are no longer refined and their local error does not contribute to the global error used in the adaptation decision. Hence, realistic minimal edge lengths stop refinement in singularities and layers.

- Decrease the accuracy requirement in adaptation criteria. Realistic relative error tolerances for the Dirichlet adaptation criteria are about 0.1 or greater. The comparison relation between Dirichlet and residual adaptation criteria results in relative error tolerances $\text{RelError} > 0.4$ for the residual criteria.
- Reduce the maximal number of elements in the macro element tree (`MaxNumberMacroElements`).

Convergence Problems After Adaptation

It has been observed that, for very coarse simulation grids, the recomputation procedure shows convergence problems. Such problems can be solved by using slightly refined initial grids or by refining the coarsest possible grid (reduce `MaxElementSize` in refinement definitions). On the other hand, the changes between consecutive grids could be too large, which can be controlled by reducing the number of tree loop iterations. Convergence problems occur in the presence of strong nonlinearities (caused by the selected transport model or certain physical models). Restricting the physical complexity may solve the problem and resulting grids may be still useful for the intended simulation case.

AGM and Extrapolate

After adaptation within a quasistationary, extrapolation is not possible and the parameter step size may decrease. Extrapolation is supported as soon as two consecutive solutions are computed on the same mesh.

References

- [1] B. Schmittüsén, *Grid Adaptation for the Stationary Two-Dimensional Drift-Diffusion Model in Semiconductor Device Simulation*, Series in Microelectronics, vol. 126, Konstanz, Germany: Hartung-Gorre, 2002.
- [2] B. Schmittüsén, K. Gärtner, and W. Fichtner, *A Grid Adaptation Procedure for the Stationary 2D Drift-Diffusion Model Based on Local Dissipation Rate Error Estimation: Part I - Background*, Technical Report 2001/02, Integrated Systems Laboratory, ETH, Zurich, Switzerland, December 2001.
- [3] B. Schmittüsén, K. Gärtner, and W. Fichtner, *A Grid Adaptation Procedure for the Stationary 2D Drift-Diffusion Model Based on Local Dissipation Rate Error Estimation: Part II - Examples*, Technical Report 2001/03, Integrated Systems Laboratory, ETH, Zurich, Switzerland, December 2001.
- [4] R. Verfürth, *A Review of A Posteriori Error Estimation and Adaptive Mesh-Refinement Techniques*, Chichester: Wiley Teubner, 1996.

32: Automatic Grid Generation and Adaptation Module AGM

References

This chapter presents some of the numeric methods used in Sentaurus Device.

Discretization

The well-known ‘box discretization’ [1][2][3] is applied to discretize the partial differential equations (PDEs). This method integrates the PDEs over a test volume such as that shown in [Figure 113](#), which applies the Gaussian theorem, and discretizes the resulting terms to a first-order approximation.

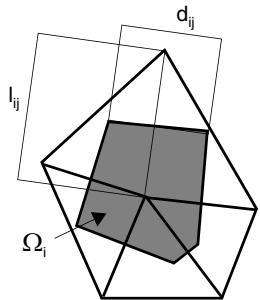


Figure 113 Single box for a triangular mesh in 2D

In general, box discretization discretizes each PDE of the form:

$$\nabla \cdot \vec{J} + R = 0 \quad (916)$$

into:

$$\sum_{j \neq i} \kappa_{ij} \cdot j_{ij} + \mu(\Omega_i) \cdot r_i = 0 \quad (917)$$

with values listed in [Table 115](#).

Table 115 Dimension

Dimension	κ_{ij}	$\mu(\Omega_i)$
1D	$1/l_{ij}$	Box length
2D	d_{ij}/l_{ij}	Box area
3D	D_{ij}/l_{ij}	Box volume

In this case, the physical parameters j_{ij} and r_i have the values listed in [Table 116](#), where $B(x) = x/(e^x - 1)$ is the Bernoulli function.

Table 116 Equations

Equation	j_{ij}	r_i
Poisson	$\epsilon(u_i - u_j)$	$-\rho_i$
Electron continuity	$\mu^n (n_i B(u_i - u_j) - n_j B(u_j - u_i))$	$R_i - G_i + \frac{d}{dt} n_i$
Hole continuity	$\mu^P (p_j B(u_j - u_i) - p_i B(u_i - u_j))$	$R_i - G_i + \frac{d}{dt} p_i$
Temperature	$\kappa(T_i - T_j)$	$H_i - \frac{d}{dt} T_i c_i$

One special feature of Sentaurus Device is that the actual assembly of the nonlinear equations is performed elementwise, that is:

$$\sum_{e \in \text{elements}(i)} \left\{ \left(\sum_{j \in \text{vertices}(e)} \kappa_{ij}^e \cdot j_{ij}^e \right) + \mu^e(\Omega_i) \cdot r_i^e \right\} = 0 \quad (918)$$

This expression is equivalent to [Eq. 917](#), but has the advantage that some parameters (such as ϵ , μ_n , μ_p) can be handled elementwise, which is useful for numeric stability and physical exactness.

Box Method Coefficients

Basic Definitions

A mesh is a Delaunay mesh if the interior of the circumsphere (circumcircle for 2D) of each element contains no mesh vertices.

Let T be a mesh element. The center circumsphere (circle for 2D) around the element T is called the element Voronoï center V_T . Let f be the face of the element T . The center circumcircle around the face f is called the face Voronoï center V_f .

Let v be a vertex of the mesh and let $ev^n (1 \leq n \leq N)$ be the set of edges connected to vertex v . Let P_{ev}^n be the mid-perpendicular plane for the edge ev^n . The plane P_{ev}^n splits 3D space into two half-spaces. Let S_{ev}^n be the half-space that contains the vertex v . The intersection of all half-spaces S_{ev}^n is called the Voronoï box B_v of vertex v . The Voronoï box B_v is the convex

polyhedron and any face of B_v is called a Voronoï face. In addition, $T_v^m (1 \leq m \leq M)$ is the set of elements per vertex v and $T_{ev}^k (1 \leq k \leq K)$ is the set of elements per edge ev .

For a Delaunay mesh, there are two propositions:

1. The vertices of a Voronoï box B_v are element Voronoï centers, that is, B_v is a polyhedron with the vertices $(V_{T_v^1}, V_{T_v^2}, \dots, V_{T_v^M})$.
2. The Voronoï face associated with the edge ev is the convex polygon with the vertices $(V_{T_{ev}^1}, V_{T_{ev}^2}, \dots, V_{T_{ev}^K})$ and this polygon lies in the mid-perpendicular plane P_{ev} (see [Figure 114](#)).

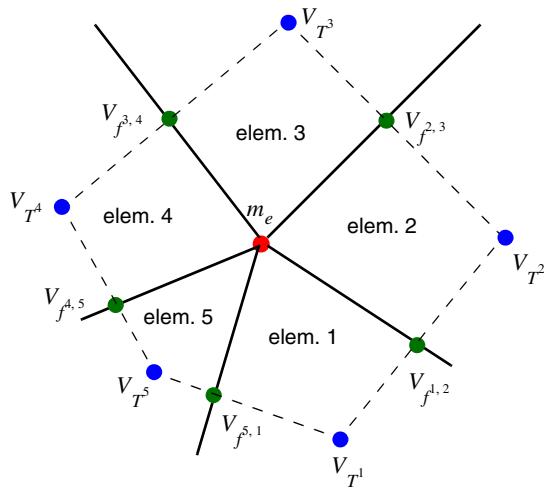


Figure 114 Voronoï face of 3D Delaunay elements: view of mid-perpendicular plane at edge e with element Voronoï centers V_{T^i} and the face Voronoï center $V_{f^{i,i+1}}$ between elements T^i and T^{i+1}

33: Numeric Methods

Box Method Coefficients

For a non-Delaunay mesh, the Voronoï box is a convex polyhedron, but there are vertices that are not Voronoï element centers (see [Figure 115](#), vertex R).

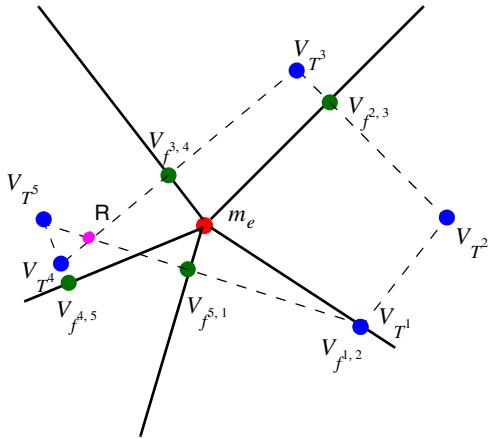


Figure 115 Voronoï face of 3D non-Delaunay elements: non-Delaunay mesh; Voronoï face is polygon $(V_{T^1}, V_{T^2}, V_{T^3}, R, V_{T^l})$

Variation of Box Method Algorithms

The various box methods differ in the way that the 2D volume of the Voronoï face is split into the elements T_{ev}^k associated with edge ev . All box methods give the same result if the mesh has no obtuse elements (an element is obtuse if its element Voronoï center is outside of the element).

Sentaurus Device implements of the following box method (BM) algorithms: element intersection BM, element-face intersection BM, and quadrilateral BM. [Figure 116 on page 889](#) shows a situation where these methods produce different results. For all methods, the Voronoï faces for elements T^1, T^2, T^3 are the same and are equal to the area of the polygons $(m_e, V_{f^{i-1,i}}, V_{T^i}, V_{f^{i+1,i}}, m_e)$. The elements T^4, T^5 have the following Voronoï faces depending on the BM algorithms:

- Element intersection BM algorithms
 - Element T^4 : area of polygon $(m_e, V_{f^{3,4}}, V_{T^4}, V_{T^5}, p, m_e)$
 - Element T^5 : area of polygon $(m_e, p, V_{f^{5,1}}, m_e)$
- Element-face intersection BM algorithms
 - Element T^4 : area of polygon $(m_e, V_{f^{3,4}}, V_{T^4}, V_{T^5}, m_e)$
 - Element T^5 : area of polygon $(m_e, V_{T^5}V_{f^{5,1}}, m_e)$

- Quadrilateral BM algorithms
 - Element T^4 : area of polygon ($m_e, V_{f^{3,4}}, V_{T^4}, V_{f^{4,5}}, m_e$)
 - Element T^5 : area of polygon ($m_e, V_{T^5}V_{f^{5,1}}, m_e$) minus area of polygon ($m_e, V_{T^5}V_{f^{4,5}}, m_e$)

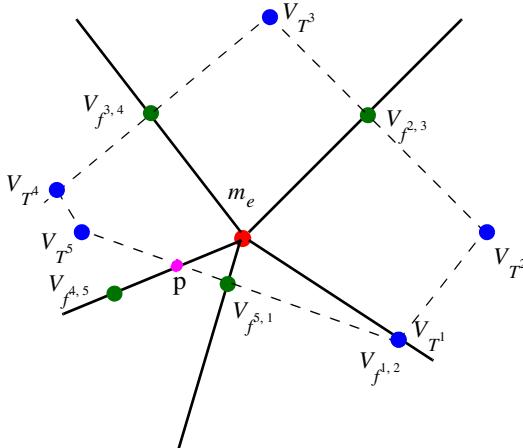


Figure 116 Voronoï face of 3D elements to consider different box method algorithms

Truncated and Non-Delaunay Elements

If an obtuse element has an obtuse face on the material or region boundary, then for this element, an algorithm of truncation can be applied. [Figure 117](#) shows the difference between the original and truncated Voronoï polygons in the 2D case.

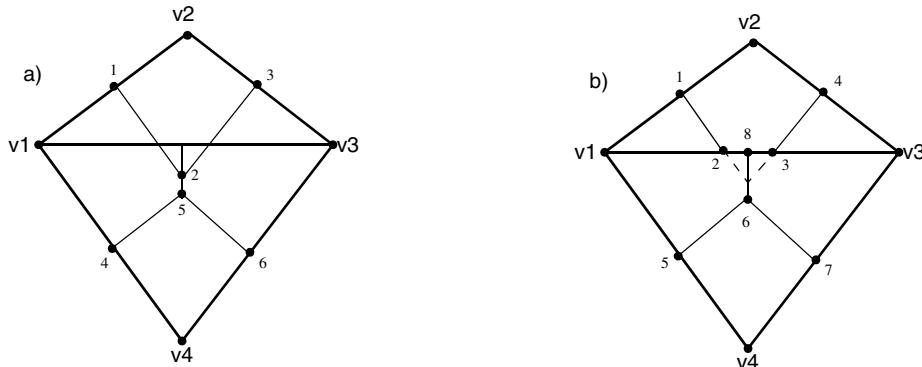


Figure 117 Box method in 2D for truncated element: (a) Voronoï polygons before truncation – $P1(v1,1,2,5,4,v1)$, $P2(v2,1,2,3,v2)$, $P3(v3,3,2,5,6,v3)$, $P4(v4,4,5,6,v4)$; (b) Voronoï polygons after truncation – $P1(v1,1,2,8,6,5,v1)$, $P2(v2,1,2,3,4,v2)$, $P3(v3,4,3,8,6,7,v3)$, $P4(v4,5,6,7,v4)$

For the 3D case, a similar algorithm of truncation is used. If an element is non-Delaunay, it is truncated in any case. In the 2D case, there is a soft truncation for the non-Delaunay element if the neighbor elements have the same material (see [Figure 118](#)).

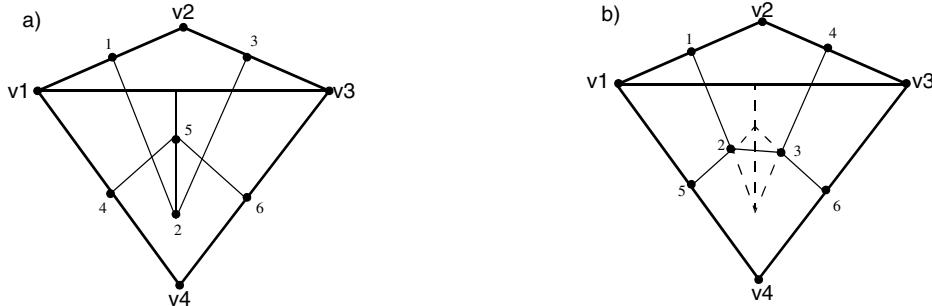


Figure 118 Box method in 2D for non-Delaunay element: (a) Voronoi polygons before truncation – $P_1(v_1, 1, 2, 5, 4, v_1)$, $P_2(v_2, 1, 2, 3, v_2)$, $P_3(v_3, 3, 2, 5, 6, v_3)$, $P_4(v_4, 4, 5, 6, v_4)$ (the polygons P_1 and P_3 are singular; the polygons P_2 and P_4 have an intersection); (b) Voronoi polygons after truncation – $P_1(v_1, 1, 2, 5, v_1)$, $P_2(v_2, 1, 2, 3, 4, v_2)$, $P_3(v_3, 3, 4, 6, v_3)$, $P_4(v_4, 5, 2, 3, 6, v_4)$ (the polygons P_1 and P_3 are not singular; the polygons P_2 and P_4 have no intersection)

Math Parameters for Box Method Coefficients

The coefficients needed for discretization $\mu^e(\Omega_i)$ and κ_{ij}^e from [Eq. 918](#) (referred to as Measure and Coefficients) can be computed inside Sentaurus Device or read from outside files.

The keyword `-AverageBoxMethod` in the Math section selects the quadrilateral box method (BM) algorithm.

`AverageBoxMethod` and `NaturalBoxMethod` in the Math section select the element intersection BM algorithms. For `AverageBoxMethod`, the algorithm is element oriented (it computes the element Voronoi face for each element and all edges of this element). For `NaturalBoxMethod`, the algorithm is edge oriented (it computes the element Voronoi faces for each edge and all elements around this edge), requires a Delaunay mesh, and can be applied only to 3D structures.

The average box method algorithm has two implementation `AverageBoxMethod` (default) and `CVPL_AverageBoxMethod` (optional). The optional implementation provides different calculations of box coefficients, which may be useful for ‘very non-Delaunay’ meshes.

`VoronoiFaceBoxMethod=<scheme>` in the `Math` section selects an element-face intersection BM algorithm, where `<scheme>` denotes the choice for the truncated correction:

- Truncated for correction for all obtuse elements.
- -Truncated for no correction.
- `RegionBoundaryTruncated` or `MaterialBoundaryTruncated` for correction of obtuse elements with an obtuse face (edge for 2D) on boundary of regions or materials.

Note that for non-Delaunay elements, the correction is applied in any case, no matter which scheme is selected.

[Table 149 on page 1112](#) lists all available options for computing `Measure` and `Coefficients`.

Only one BM algorithm can be activated. The default options are:

```
Math { ...
    AverageBoxMethod -VoronoiFaceBoxMethod -NaturalBoxMethod
    BoxMethodFromFile -BoxCoefficientsFromFile -BoxMeasureFromFile
    -CVPL_AverageBoxMethod
    ...
}
```

Saving and Restoring Box Method Coefficients

Usually, the coefficients needed for discretization are computed inside Sentaurus Device. For experimental purposes, it may be preferred to use externally provided data. `Measure` and `Coefficients` ($\mu^e(\Omega_i)$ and κ_{ij}^e from [Eq. 918, p. 886](#)) can be stored in, and loaded into and from the debug file. There are two options for element numbering in such files:

- Internal Sentaurus Device numbering with `MeasureCoefficientsDebug` as the debug file name.
- Mesh numbering (from grid file) with `MeasureCoefficients.debug` as the debug file name for this option.

If the keyword `BoxMeasureFromFile` or `BoxCoefficientsFromFile` is specified in the `Math` section and there is file `MeasureCoefficientsDebug` in the simulation directory, Sentaurus Device reads the corresponding arrays from this file.

If the keyword `BoxMeasureFromFile(GrdNumbering)` or the keyword `BoxCoefficientsFromFile(GrdNumbering)` is specified and there is the file `MeasureCoefficients.debug`, Sentaurus Device reads the corresponding arrays from this file. If there are no such debug files but these keywords are specified, Sentaurus Device computes `Measure` and `Coefficients` and writes them in the corresponding file.

33: Numeric Methods

Box Method Coefficients

The format of the `MeasureCoefficientsDebug` file is as follows. In line k of the `Measure` section, the control volume for each element-vertex j of element k is stored (that is, value `Measure [k] [j]`). The enumeration of elements and local enumeration of vertices inside the element (see [Figure 120 on page 926](#)) correspond to the internal Sentaurus Device numbering.

The `Coefficients` section in this file has a similar format. For example, the `Measure` section in the file can appear as follows:

```
Measure {
    8.719666833501378e-08 4.359833416750702e-08 4.359833416750729e-08
    8.719666833501378e-08 4.359833416750702e-08 4.359833416750729e-08
    ...
}
```

The format of the `MeasureCoefficients.debug` file is different. There are four section: `Info`, `Elem_type`, `Measure`, and `Coefficients`. In line k of the `Measure` section, the control volume for each element-vertex j of element k is stored (that is, value `Measure [k] [j]`). The enumeration of elements and local enumeration of vertices inside the element correspond to the grid file (see [Utilities User Guide, Figure 7 on page 24](#)). The `Coefficients` section in the debug file has similar format. For example, the file can look like:

```
Info {
    dimension      = 2
    nb_vertices    = 10
    nb_grd_elements = 11
    nb_des_elements = 7
}

Elem_type {
    point        = 0
    line         = 1
    triangle     = 2
    rectangle    = 3
    tetrahedron  = 5
    pyramid      = 6
    prism        = 7
    cuboid       = 8
}

Measure { # unit = [um^2]
# grd_elem des_elem elem_type
    0 0 2 1.828427124999999e+00 9.14213562500004e-01 9.14213562500002e-01
    1 1 2 4.052251462735666e+00 4.052251462735666e+00 8.104502925471332e+00
    ...
    7 -1 1      # contact or interface
    8 -1 1      # contact or interface
    ...
}
```

```

}

Coefficients { # unit = [1]
# grd_elem des_elem elem_type
0 0 2 1.093836321204215e+00 1.100111438811216e-16 2.285533906249999e-01
1 1 2 0.000000000000000e+00 8.379715512271076e-01 8.379715512271076e-01
...
7 -1 1      # contact or interface
8 -1 1      # contact or interface
...
}

```

AC Simulation

AC simulation is based on small-signal AC analysis. The response of the device to ‘small’ sinusoidal signals superimposed upon an established DC bias is computed as a function of frequency and DC operating point. Steady-state solution is used to build up a linear algebraic system [4] whose solution provides the real and imaginary parts of the variation of the solution vector (ϕ, n, p, T_n, T_p, T) induced by small sinusoidal perturbation at the contacts.

AC Response

The AC response is obtained from the three basic semiconductor equations (see [Eq. 26, p. 178](#) and [Eq. 27, p. 178](#)) and from up to three additional energy conservation equations to account for electron, hole, and lattice temperature responses. In the following description of the AC system, the temperatures have been omitted in the solution vector and Jacobian for simplicity, a complete description being formally obtained by adding the temperature responses to the solution vector and the corresponding lines to the system Jacobian.

After discretization, the simplified system of equations can be symbolically represented at the node i of the computation mesh as:

$$F_{\phi i}(\phi, n, p) = 0 \quad (919)$$

$$F_{ni}(\phi, n, p) = G_{ni}(n) \quad (920)$$

$$F_{pi}(\phi, n, p) = \dot{G}_{pi}(p) \quad (921)$$

where F and G are nonlinear functions of the vector arguments ϕ, n, p , and the dot denotes time differentiation.

33: Numeric Methods

AC Simulation

By substituting the vector functions of the form $\xi_{\text{total}} = \xi_{\text{DC}} + \tilde{\xi}e^{i\omega t}$ into [Eq. 919](#), [Eq. 920](#), and [Eq. 921](#) where $\xi = \phi, n, p$, ξ_{DC} is the value of ξ at the DC operating point, and $\tilde{\xi}$ is the corresponding response (or the phasor uniquely identifying the complex perturbation) and then expanding the nonlinear functions F and G in the Taylor's series around the DC operating point and keeping only the first-order terms (the small-signal approximation), the AC system of equations at the node i can be written as:

$$\sum_j \begin{bmatrix} \frac{\partial F_{\phi i}}{\partial \phi_j} & \frac{\partial F_{\phi i}}{\partial n_j} & \frac{\partial F_{\phi i}}{\partial p_j} \\ \frac{\partial F_{ni}}{\partial \phi_j} \frac{\partial F_{ni}}{\partial n_j} - i\omega \frac{\partial G_{ni}}{\partial n_j} & \frac{\partial F_{ni}}{\partial p_j} \\ \frac{\partial F_{pi}}{\partial \phi_j} & \frac{\partial F_{pi}}{\partial n_j} & \frac{\partial F_{pi}}{\partial p_j} - i\omega \frac{\partial G_{pi}}{\partial p_j} \end{bmatrix}_{\text{DC}} \begin{bmatrix} \tilde{\phi}_j \\ \tilde{n}_j \\ \tilde{p}_j \end{bmatrix} = 0 \quad (922)$$

where the solution vector is scaled with respect to terminal voltages (at the contact where the voltage is applied, ϕ is 1). Therefore, the unit of carrier density responses is $\text{cm}^{-3}\text{V}^{-1}$ and the potential response is unitless.

The matrix of [Eq. 922](#) differs from the Jacobian of the system of equations [Eq. 919](#), [Eq. 920](#), and [Eq. 921](#) only by pure imaginary additive terms involving derivatives of G with respect to carrier densities. The global AC matrix system is obtained by imposing the corresponding AC boundary conditions and performing the summation (assembling the global matrix).

Common AC boundary conditions used in AC simulation are Neumann boundary and oxide–semiconductor jump conditions carried over directly from DC simulation; Dirichlet boundary conditions for carrier densities where n and p at Ohmic contacts are $\tilde{n} = \tilde{p} = 0$; and Dirichlet boundary conditions for AC potential at Ohmic contacts that are used to excite the system.

After assembling the global AC matrix and taking into account the boundary conditions, the AC system becomes:

$$[J + iD]\tilde{X} = B \quad (923)$$

where J is the Jacobian matrix, D contains the contributions of the G functions to the matrix, B is a real vector dependent on the AC voltage drive, and \tilde{X} is the AC solution vector.

By writing the solution vector as $\tilde{X} = X_R + iX_I$ with X_R and X_I the real and imaginary part of the solution vector respectively, the AC system can be rewritten using only real arithmetic as:

$$\begin{bmatrix} J & -D \\ D & J \end{bmatrix} \begin{bmatrix} X_R \\ X_I \end{bmatrix} = \begin{bmatrix} B \\ 0 \end{bmatrix} \quad (924)$$

The AC response is actually computed by solving the real system Eq. 924.

An ACPlot statement in the System section is used to plot AC responses $(\tilde{\phi}, \tilde{n}, \tilde{p}, \tilde{T}_n, \tilde{T}_p, \tilde{T})$. The responses are plotted in the AC plot file of Sentaurus Device with a separate file for each frequency.

For details of the ACPlot statement, see [Table 129 on page 1097](#). For details and examples of small-signal AC analysis, see [ACCoupled: Small-Signal AC Analysis on page 161](#).

AC Current Density Responses

When the AC system is solved, the AC current density responses $\dot{\tilde{J}}_D$, $\dot{\tilde{J}}_n$, and $\dot{\tilde{J}}_p$ are computed using:

$$\dot{\tilde{J}}_D = -i\omega\epsilon\nabla\tilde{\phi} \quad (925)$$

$$\dot{\tilde{J}}_n = \left. \frac{\partial \dot{\tilde{J}}_n}{\partial \phi} \right|_{DC} \tilde{\phi} + \left. \frac{\partial \dot{\tilde{J}}_n}{\partial n} \right|_{DC} \tilde{n} + \left. \frac{\partial \dot{\tilde{J}}_n}{\partial p} \right|_{DC} \tilde{p} \quad (926)$$

$$\dot{\tilde{J}}_p = \left. \frac{\partial \dot{\tilde{J}}_p}{\partial \phi} \right|_{DC} \tilde{\phi} + \left. \frac{\partial \dot{\tilde{J}}_p}{\partial p} \right|_{DC} \tilde{p} + \left. \frac{\partial \dot{\tilde{J}}_p}{\partial n} \right|_{DC} \tilde{n} \quad (927)$$

The unit of current density responses is $\text{Acm}^{-2}\text{V}^{-1}$.

The responses of the heat fluxes for the lattice ($\dot{\tilde{S}}_L$), electrons ($\dot{\tilde{S}}_n$), and holes ($\dot{\tilde{S}}_p$) are analogous. Their unit is $\text{Wcm}^{-2}\text{V}^{-1}$.

The ACPlot statement in the System section is used to plot the AC current density responses. The responses are added to the AC solution response in the AC plot files of Sentaurus Device.

Harmonic Balance Analysis

Harmonic balance (HB) analysis is a frequency domain method to solve periodic and quasi-periodic time-dependent problems for steady-state solutions [5][6]. It is a popular method for RF circuit design applications. While transient discretization schemes allow the simulation of arbitrary time-dependent problems, HB more efficiently models periodic and quasi-periodic problems for systems with time constants that vary by many orders of magnitude. The detailed command file syntax is given [Harmonic Balance on page 164](#).

Harmonic Balance Equation

In general, the dynamic mixed-mode simulation problem takes the form:

$$\frac{d}{dt}q[r, u(t, r)] + f[r, u(t, r), w(t)] = 0 \quad (928)$$

where f and q are nonlinear functions, w represents explicitly time-dependent devices (in particular, voltage or current sources), and the function u is the vector of all solution variables.

Let f_1, \dots, f_K be a set of different frequencies with $f_k > 0$, then both the sources and the solution are approximated by a truncated Fourier series:

$$u(t) = U_0 + \sum_{1 \leq k \leq K} \{U_k \exp(i\omega_k t) + U_k^* \exp(-i\omega_k t)\} \quad (929)$$

A formal Fourier transform of Eq. 928 results in the HB equation for the problem:

$$L(U) = i\Omega Q(U) + F(U) = 0 \quad (930)$$

where F and Q are the finite Fourier series of f and q , respectively, Ω is the frequency matrix, and U is the vector of all Fourier coefficients of u .

Multitone Harmonic Balance Analysis

The multitone harmonic balance (HB) analysis makes use of the multidimensional Fourier transformation (MDFT). This means that the problem is mapped onto a problem in a multidimensional frequency and multidimensional time domain, hereby exploiting the equivalence of Fourier spectra of quasi-periodic functions with their corresponding multidimensional functions.

Multidimensional Fourier Transformation

The multidimensional Fourier transformation (MDFT) maps multidimensional functions onto a multidimensional spectrum.

Let M be a positive integer, the number of tones, $\hat{f}_1, \dots, \hat{f}_M$, be a finite set of different positive numbers, the base frequencies of tones, and H_1, \dots, H_M nonnegative integer numbers, the maximal number of harmonics for each tone.

Define for each tone m the m -th base period $T_m := 1/\hat{f}_m$, the m -th circular frequency $\omega_m := 2\pi\hat{f}_m$, the m -th (minimum) number of sampling points $S_m := 2H_m + 1$, and the m -th (maximum) sampling interval $\delta_m = T_m/S_m$.

Furthermore, let $\underline{x} = (x_1, \dots, x_M)^T$ denote the M -dimensional vector, and let $D_{\underline{x}} = \text{diag}(x_1, \dots, x_M)$ denote the $M \times M$ -matrix composed of the values x_1, \dots, x_M .

The set of multi-indices associated with \underline{H} is given by:

$$K := \{\underline{h} \in \mathbf{Z}^M : -H_m \leq h_m \leq H_m \text{ for all } 1 \leq m \leq M\} \quad (931)$$

Let u^M be a function on the M -dimensional space \mathbf{C}^M given by:

$$u^M(t_1, \dots, t_M) = \sum_{\underline{h} \in K} U_{\underline{h}}^M \exp(i\underline{h} D_{\underline{\omega}} \underline{t}) \quad (932)$$

with given complex numbers $U_{\underline{h}}^M$, then:

$$U_{\underline{h}}^M = \frac{1}{T} \int_0^{T_1} \dots \int_0^{T_M} u^M(t_1, \dots, t_M) \exp(-i\underline{h} D_{\underline{\omega}} \underline{t}) dt_1 \dots dt_M \quad (933)$$

with $T := \prod_{1 \leq m \leq M} T_m$. $U^M = (U_{\underline{h}}^M)_{\underline{h} \in K}$ is the multidimensional spectrum of u^M . Sampling the function in all dimensions at the equidistant sampling points $t_{\underline{s}} = D_{\underline{\delta}} \underline{s}$ ($0 \leq \underline{s} < \underline{S}$), the discrete MDFT is written formally as:

$$U^M = \Gamma^M u^M \quad (934)$$

that is, Γ^M is a linear map from \mathbf{C}^S onto \mathbf{C}^S where $S := \prod_{1 \leq m \leq M} S_m$ is the total number of sampling points.

Quasi-Periodic Functions

The multidimensional function u^M can be projected onto a one-dimensional time space by:

$$u(t) := u^M(t, \dots, t) = \sum_{\underline{h} \in K} U_{\underline{h}} \exp(i\underline{h} \underline{\omega} \underline{t}) \quad (935)$$

33: Numeric Methods

Harmonic Balance Analysis

Functions satisfying this representation are called quasi-periodic. The set:

$$\Lambda := \{f_{\underline{h}} \in \mathbf{R} : f_{\underline{h}} = \underline{h} \cdot \hat{f} \text{ for all } \underline{h} \in K\} \quad (936)$$

is the spectrum domain associated with f and K (or H). The projection is invertible if, for two different multi-indices \underline{h}_1 and \underline{h}_2 in K , the resulting frequencies $f_{\underline{h}_1}$ and $f_{\underline{h}_2}$ are different. Note that the one-dimensional Fourier spectrum of u coincides with the multidimensional spectrum of u^M .

While for the multidimensional function u^M S sample points can be specified to compute the multidimensional spectrum, the one-dimensional sample points for u are not well defined (but are rather virtual in the multidimensional time domain).

Multidimensional Frequency Domain Problem

The multitone HB analysis is essentially a translation of (one-dimensional or multidimensional) time-domain problems in a multidimensional frequency domain. Though originally derived from a time-domain problem, the circuit equations are directly specified in a multidimensional frequency domain. This avoids sampling of (one-dimensional) time-dependent sources, which cannot be performed accurately on a sample set of size S . This is the reason why the compact circuit models must provide the CMI-HB-MDFT function set.

The Fourier transformation of quasi-periodic functions is the composition:

$$\Gamma = \Gamma^M \circ P^{-1} \quad (937)$$

where Γ^M is the multidimensional Fourier transformation of Eq. 934 and P^{-1} is the inverse of the projection Eq. 935.

One-Tone Harmonic Balance Analysis

For one-tone HB analysis, the standard discrete Fourier transformation can be used, which includes that the sampling points are defined explicitly in a (one-dimensional) time domain. Therefore, the problem can be extracted directly from the time-domain formulation of the circuit.

Solving HB Equation

The HB equation (Eq. 930) is a nonlinear equation in U and is solved by the Newton algorithm. In each Newton step, the linear equation:

$$\frac{\partial L}{\partial U}(U) \cdot \delta U = -L(U) \quad (938)$$

must be solved.

The Jacobian $\partial L / \partial U$ in Fourier space is computed from the Jacobian in the time domain as follows: For a nonlinear scalar function $g: \mathbf{R} \rightarrow \mathbf{R}$ and a T -periodic scalar signal $u(t)$, the Fourier coefficients $G \in \mathbf{C}^S$ of $g(u(t))$ are approximated:

$$G(U) = \Gamma g(\hat{u}) = \Gamma g(\Gamma^{-1}U) \quad (939)$$

where Γ and Γ^{-1} are the discrete Fourier transform operator and its inverse, \hat{u} is the vector of the time samples $u(t_i)$, and $g(\hat{u})$ is the vector of values $g(u(t_i))$.

The derivatives of the k -th Fourier component G_k with respect to the j -th Fourier component U_j read:

$$\frac{\partial G_k}{\partial U_j}(U) = \sum_l \Gamma_{kl} \frac{\partial g}{\partial U_j}(\hat{u}_l) = \sum_l \Gamma_{kl} \frac{\partial g}{\partial u}(\hat{u}_l) \Gamma_{lj}^{-1} \quad (940)$$

The corresponding Jacobian is written in the compact form:

$$\frac{\partial G}{\partial U} = \Gamma \hat{J}_u \Gamma^{-1} \text{ with } \hat{J}_u = \frac{\partial g}{\partial u}(\hat{u}) \quad (941)$$

For scalar functions g and u , \hat{J}_u is a diagonal matrix; for vector-valued functions g and u , \hat{J}_u is a block-diagonal matrix.

Using the notation above, Eq. 930 becomes:

$$L(U) = i\Omega \Gamma q(\hat{u}(U)) + \Gamma f(\hat{u}(U)) = 0 \quad (942)$$

and Eq. 938 for the Newton step takes the form:

$$(i\Omega \Gamma \hat{J}_q \Gamma^{-1} + \Gamma \hat{J}_f \Gamma^{-1}) \delta U = -L(U) \quad (943)$$

The Newton algorithm constructs a sequence U^k of the Fourier coefficients of the time-domain solution vector u . The sequence is regarded as converged if both the residual $|L(U^k)|$ and the update error are small.

Solving HB Newton Step Equation

The memory requirements for storing the HB Jacobian matrix typically become very large, as its size is increased by a factor of S^2 compared to the corresponding DC or transient matrix. For a very small number of harmonics and a moderately sized simulation grid, using a direct linear solver may be feasible. However, the use of the GMRES(m) iterative method is recommended for most applications.

Restarted GMRES Method

The HB module makes use of a preconditioned restarted generalized minimum residual GMRES(m) method [7], a Krylow subspace method, which does not need to store the Jacobian in memory, as only matrix-vector products have to be computed.

GMRES(m) requires a suitable preconditioner to achieve convergence. A (left) preconditioner P is a matrix that approximates a given matrix A , but is much easier to invert than A itself. Instead of solving the linear equation $Ax = b$ for given A and b , the (left) preconditioned problem $P^{-1}Ax = P^{-1}b$ is solved. The preconditioner used for HB [8] takes the form:

$$P = i\Omega \begin{bmatrix} \bar{J}_q & 0 \\ \dots & \dots \\ 0 & \bar{J}_q \end{bmatrix} + \begin{bmatrix} \bar{J}_f & 0 \\ \dots & \dots \\ 0 & \bar{J}_f \end{bmatrix} \quad (944)$$

where the matrix \bar{J}_f , and similarly \bar{J}_q , is computed as:

$$\bar{J}_f = \frac{1}{S} \sum_{0 \leq s \leq S-1} J_f(t_s) \quad (945)$$

and J_f denotes the Jacobian of f with respect to u . The preconditioner equals the HB Jacobian in the limit of small signals, where the coupling terms between the frequencies vanish. Therefore, each diagonal block of P corresponds to the AC matrix for the respective harmonic. This preconditioner is well suited to ‘moderately large’ signal applications.

The preconditioner can be computed without an explicit Fourier transform, and its inversion is more economical than for the full Jacobian. The inversion is performed by applying a complex direct solver for each harmonic component separately, thereby requiring the computational costs of solving $(S+1)/2$ complex-valued linear systems. The computational complexity is of the order $O(S)$ for inverting the preconditioner, and $O(S \ln(S))$ for one complete iteration step of the iterative solver, while the number of iterations necessary to achieve convergence is unknown (but bounded).

Direct Solver Method

For the direct solver, the complex-valued $S \times S$ linear system (Eq. 943) is transformed to a $S \times S$ real-valued problem, which is possible as only real-valued functions are involved. The resulting linear system is solved by the direct solver PARDISO. The direct solver requires the entire matrix stored in memory. Therefore, the memory capacity is easily exceeded for increasing S . Additionally, the computational complexity is of the order $O(S^3)$.

Transient Simulation

Transient equations used in semiconductor device models and circuit analysis can be formally written as a set of ordinary differential equations:

$$\frac{d}{dt}q(z(t)) + f(t, z(t)) = 0 \quad (946)$$

which can be mapped to the DC and transient parts of the PDEs.

Sentaurus Device uses implicit discretization of transient equations (see Eq. 946) and supports two discretization schemes: simple backward Euler (BE) and composite trapezoidal rule/backward differentiation formula (TRBDF), which is the default.

Backward Euler Method

Backward Euler is a very stable method, but it has only a first-order of approximation over time-step h_n . The discretization can be written as:

$$q(t_n + h_n) + h_n f(t_n + h_n) = q(t_n) \quad (947)$$

The local truncation error (LTE) estimation is based on the comparison of the obtained solution $q(t_n + h_n)$ with the linear extrapolation from the previous time-step. The extrapolated solution is written as:

$$q^{\text{extr}} = q(t_n) - \frac{f(t_n) + f(t_n + h_n)}{2} h_n \quad (948)$$

Then, in every point, the relative error can be estimated as $(q(t_n + h_n) - q^{\text{extr}})/q(t_n + h_n)$.

33: Numeric Methods

Transient Simulation

Using [Eq. 947](#) and [Eq. 948](#), and estimating the norm of relative error, Sentaurus Device computes the value:

$$r = \sqrt{\frac{1}{N} \sum_{i=1}^N \left(\frac{f(t_n + h_n) - f(t_n)}{\varepsilon_{R,tr} |q_n(t_n + h_n)| + \varepsilon_{A,tr} h_n} \right)^2} \quad (949)$$

where the sum is taken over all unknowns (that is, all free vertices of all equations), and $\varepsilon_{R,tr}$ and $\varepsilon_{A,tr}$ are the relative and absolute transient errors, respectively.

The next time-step is estimated as:

$$h_{\text{est}} = h_n r^{-1/2} \quad (950)$$

The value of the estimated time-step is used for h_{n+1} computation (see [Controlling Transient Simulations on page 903](#)).

TRBDF Composite Method

The transient scheme [9] for the approximation of [Eq. 946](#) is briefly reviewed in this section. From each time point t_n , the next time point $t_n + h_n$ (h_n is the current step size) is not directly reached. Instead, a step in between to $t_n + \gamma h_n$ is made. This improves the accuracy of the method. $\gamma = 2 - \sqrt{2}$ has been shown to be the optimal value. Using this, two nonlinear systems are reached.

For the trapezoidal rule (TR) step:

$$2q(t_n + \gamma h_n) + \gamma h_n f(t_n + \gamma h_n) = 2q(t_n) - \gamma h_n f(t_n) \quad (951)$$

and for the BDF2 step:

$$(2 - \gamma)q(t_n + h_n) + (1 - \gamma)h_n f(t_n + h_n) = (1/\gamma)(q(t_n + \gamma h_n) - (1 - \gamma)^2 q(t_n)) \quad (952)$$

The local truncation error (LTE) is estimated after such a double step as:

$$\tau = \left[\frac{f(t_n)}{\gamma} - \frac{f(t_n + \gamma h_n)}{\gamma(1 - \gamma)} + \frac{f(t_n + h_n)}{1 - \gamma} \right] \quad (953)$$

$$C = \frac{-3\gamma^2 + 4\gamma - 2}{12(2 - \gamma)} \quad (954)$$

Sentaurus Device then computes the following value from this:

$$r = \sqrt{\frac{1}{N} \sum_{i=1}^N \left(\frac{\tau_i}{\varepsilon_{R,tr} |q_n(t_n + h_n)| + \varepsilon_{A,tr}} \right)^2} \quad (955)$$

where the sum is taken over all unknowns (that is, all free vertices of all equations), and $\varepsilon_{R,tr}$ and $\varepsilon_{A,tr}$ are the relative and absolute transient errors, respectively. Since the TRBDF method has a second-order approximation over h_n , the next step can be estimated as:

$$h_{\text{est}} = h_n r^{-1/3} \quad (956)$$

The value of the estimated time-step is used for h_{n+1} computation (see [Controlling Transient Simulations on page 903](#)).

Controlling Transient Simulations

By default, Sentaurus Device uses the TRBDF method. To switch to backward Euler (BE), the statement `Transient=BE` must be specified in the `Math` section.

To evaluate whether a time-step was successful and to provide an estimate for the next step size, the following rules are applied:

- If one of the nonlinear systems cannot be solved, the step is refused and tried again with $h_n = 0.5 \cdot h_n$.
- Otherwise, the inequality $r < 2f_{\text{rej}}$ is tested. If it is fulfilled, the transient simulation proceeds with $h_{n+1} = h_{\text{est}}$. Otherwise, the step is re-tried with $h_n = 0.9 \cdot h_{\text{est}}$.
- The LTE is checked only if the `CheckErrorTransient` option is selected; otherwise, the selection of the next time-step is based only on convergence of nonlinear iterations.

To activate LTE evaluation and time-step control, `CheckErrorTransient` must be specified either globally (in the `Math` section) or locally as an option in the `Transient` statement. The keyword `NoCheckErrorTransient` disables time-step control. The value of the relative error is defined by the parameter `TransientDigits` according to [Eq. 10, p. 94](#).

Absolute error is given by the keyword `TransientError` or recomputed from `TransientErrRef` ($x_{\text{ref,tr}}$) using [Eq. 11, p. 94](#) (if `RelErrControl` is switched on). Sentaurus Device provides the default values of $\varepsilon_{R,tr}$, $\varepsilon_{A,tr}$, and $x_{\text{ref,tr}}$. The coefficient f_{rej} is equal to 1 by default. You can define the values of $\varepsilon_{R,tr}$, $\varepsilon_{A,tr}$, $x_{\text{ref,tr}}$, and f_{rej} globally in the `Math` section, or specify them as options in the `Transient` statement. In the latter case, it overwrites the default and `Math` specifications for this command.

Nonlinear Solvers

In the next two sections, the `Digits` variable corresponds to the keyword `Digits`, which can be given in the `Math` section (see [Nonlinear Solver-oriented Math Keywords on page 97](#)), or in parentheses of each `Plugin` or `Coupled` statement.

Fully Coupled Solution

For the solution of nonlinear systems, the scheme developed by Bank and Rose [10] is applied. This scheme tries to solve the nonlinear system $g(z) = 0$ by the Newton method:

$$\dot{g} + \dot{g}'\dot{x} = 0 \quad (957)$$

$$\dot{z}^j - \dot{z}^{j+1} = \lambda \dot{x} \quad (958)$$

where λ is selected such that $\|g_{k+1}\| < \|g_k\|$, but is as close as possible to 1. Sentaurus Device handles the error by computing an error function that can be defined by two methods.

The Newton iterations stop if the convergence criteria are fulfilled. One convergence criterion is the norm of the right-hand side, that is, $\|g\|$ in Eq. 957. Another natural criterion may be the relative error of the variables measured, such as $\left\| \frac{(\lambda x)}{z} \right\|$.

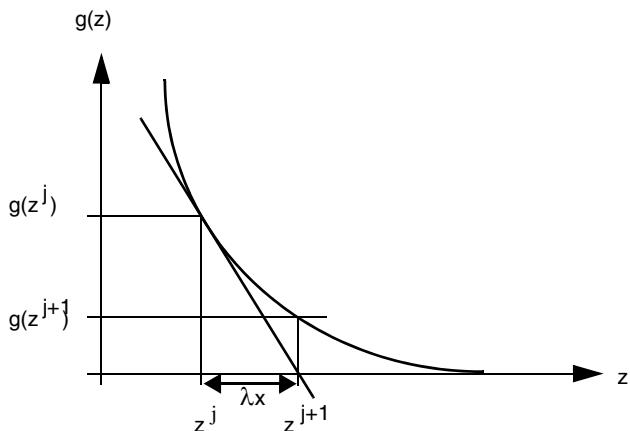


Figure 119 Newton iteration

Conversely, for very small z updates, λx must be measured with respect to some reference value of the variable z_{ref} . The formula used in Sentaurus Device as the second convergence criterion is:

$$\frac{1}{\varepsilon_R} \frac{1}{N} \sum_{e,i} \frac{|z(e,i,j) - z(e,i,j-1)|}{|z(e,i,j)| + z_{\text{ref}}(e)} < 1 \quad (959)$$

where $z(e,i,j)$ is the solution of the equation e (Poisson, electron, hole, and so on) at node i after Newton iteration j . The constant N is given by the total number of nodes multiplied by the total number of equations. The parameter ε_R is the relative error criterion.

The value of $\varepsilon_R = 10^{-\text{Digits}}$ is set by specifying the following in the Math section:

```
Math{...
    Digits = 5
}
```

where 5 is the default for Digits. The reference values $z_{\text{ref}}(e)$ ensure numeric stability even for cases when $z(e,i,j)$ is zero or very small. This error condition ensures that the respective equations are solved to an accuracy of approximately $z_{\text{ref}}(e)\varepsilon_R$.

[Eq. 959](#) can be written in the symbolic form:

$$\frac{1}{\varepsilon_R} \left\| \frac{\lambda x}{z^j + z_{\text{ref}}} \right\| < 1 \quad (960)$$

[Eq. 960](#) can also be rewritten in the equivalent form:

$$\left\| \frac{\lambda \bar{x}}{\varepsilon_R \bar{z}^j + \varepsilon_A} \right\| < 1 \quad (961)$$

where $\bar{z}^j = z^j/z^*$ and $\bar{x} = x/z^*$.

z^* is the normalization factor (for example, it is the intrinsic carrier density $n_i = 1.48 \times 10^{10} \text{ cm}^{-3}$ for electron and hole equations, and the thermal voltage $u_{T0} = 25.8 \text{ mV}$ for the Poisson equation).

The absolute error is related to the relative error through:

$$\varepsilon_A = \varepsilon_R \frac{z_{\text{ref}}}{z^*} \quad (962)$$

33: Numeric Methods

Nonlinear Solvers

Sentaurus Device supports two schemes for controlling the error conditions. The default scheme is based on [Eq. 961](#). The default values for the parameters ε_A are given in [Table 133 on page 1100](#). They are accessible in the Math section:

```
Math{...
    Error( Electron ) = 1e-5
    Error( Hole )      = 1e-5
}
```

The second scheme is activated with the keyword `RelErrControl` in the Math section and is based on [Eq. 959](#). The default values for the parameters z_{ref} are given in [Table 133 on page 1100](#).

They are accessible in the Math section:

```
Math{...
    RelErrControl
    ErrRef( Electron ) = 1e10
    ErrRef( Hole )      = 1e10
}
```

NOTE The use of the keyword `RelErrControl` is recommended, even if none of the z_{ref} parameters are actually redefined.

‘Plugin’ Iterations

This is the traditional scheme, which is also known as ‘Gummel iterations’ in most other device simulators. Consider that there are n sets of nonlinear systems $g_j(z_1 \dots z_n) = 0$. (n can be, for example, 3 and the sets can be the Poisson equation and two continuity equations.) This method starts with values $z_1^{(1)}, \dots, z_n^{(1)}$ and then solves each set $g_j = 0$ separately and consecutively. One loop could be:

$$\begin{aligned} g_1(z_1 z_2^{(i)} \dots z_n^{(i)}) &= 0 \Rightarrow z_1^{(i+1)} \\ \dots \\ g_1(z_1^{(i+1)} \dots z_{n-1}^{(i+1)} z_n) &= 0 \Rightarrow z_n^{(i+1)} \end{aligned} \tag{963}$$

If an update (λx) of the solution between two successive plugin iterations is defined as:

$$(\lambda x) = z_j^{(i+1)} - z_j^{(i)} \tag{964}$$

[Eq. 960](#) or [Eq. 961](#) can be applied for convergence control in plugin iterations.

References

- [1] R. E. Bank, D. J. Rose, and W. Fichtner, "Numerical Methods for Semiconductor Device Simulation," *IEEE Transactions on Electron Devices*, vol. ED-30, no. 9, pp. 1031–1041, 1983.
- [2] R. S. Varga, *Matrix Iterative Analysis*, Englewood Cliffs, New Jersey: Prentice-Hall, 1962.
- [3] E. M. Buturla *et al.*, "Finite-Element Analysis of Semiconductor Devices: The FIELDAY Program," *IBM Journal of Research and Development*, vol. 25, no. 4, pp. 218–231, 1981.
- [4] S. E. Laux, "Application of Sinusoidal Steady-State Analysis to Numerical Device Simulation," in *New Problems and New Solutions for Device and Process Modelling: An International Short Course held in association with the NASECODE IV Conference*, Dublin, Ireland, pp. 60–71, 1985.
- [5] B. Troyanovsky, Z. Yu, and R. W. Dutton, "Physics-based simulation of nonlinear distortion in semiconductor devices using the harmonic balance method," *Computer Methods in Applied Mechanics and Engineering*, vol. 181, no. 4, pp. 467–482, 2000.
- [6] P. J. C. Rodrigues, *Computer-Aided Analysis of Nonlinear Microwave Circuits*, Boston: Artech House, 1998.
- [7] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Philadelphia: SIAM, 2nd ed., 2003.
- [8] P. Feldmann, B. Melville, and D. Long, "Efficient Frequency Domain Analysis of Large Nonlinear Analog Circuits," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, San Diego, CA, USA, pp. 461–464, May 1996.
- [9] R. E. Bank *et al.*, "Transient Simulation of Silicon Devices and Circuits," *IEEE Transactions on Computer-Aided Design*, vol. CAD-4, no. 4, pp. 436–451, 1985.
- [10] R. E. Bank and D. J. Rose, "Global Approximate Newton Methods," *Numerische Mathematik*, vol. 37, no. 2, pp. 279–295, 1981.

33: Numeric Methods

References

Part V Physical Model Interface

This part of the Sentaurus Device manual contains the following chapter:

[Chapter 34 Physical Model Interface on page 911](#)

This chapter discusses the flexible interface that is used to add new physical models to Sentaurus Device.

Overview

The physical model interface (PMI) provides direct access to certain models in the semiconductor transport equations. You can provide new C++ functions to compute these models, and Sentaurus Device loads the functions at run-time using the dynamic loader. No access to the Sentaurus Device source code is necessary. You can modify the following models:

- Generation–recombination rate R_{net} , compared to [Eq. 27, p. 178](#)
- Avalanche generation, that is, ionization coefficient α in [Eq. 307, p. 321](#)
- Electron and hole mobilities μ_n and μ_p , compared to [Eq. 28, p. 179](#) and [Eq. 29, p. 179](#)
- Band gap, see [Chapter 5 on page 225](#)
- Bandgap narrowing E_{bgn} , see [Band Gap and Electron Affinity on page 225](#)
- Electron affinity, see [Band Gap and Electron Affinity on page 225](#)
- Apparent band-edge shift, see [Density Gradient Quantization Model on page 260](#)
- Multistate configuration–dependent apparent band-edge shift, see [Apparent Band-Edge Shift on page 368](#)
- Effective mass, see [Effective Masses and Effective Density-of-States on page 237](#)
- Energy relaxation times τ , compared to [Eq. 51, p. 187](#) to [Eq. 53, p. 187](#)
- Lifetimes τ , as used in SRH recombination (see [Eq. 260, p. 292](#)) and CDL recombination (see [Eq. 289, p. 315](#))
- Thermal conductivity κ , compared to [Eq. 32, p. 180](#)
- Heat capacity c_L , compared to [Eq. 32, p. 180](#)
- Optical absorption, see [Optical Absorption on page 998](#)
- Refractive index, see [Refractive Index Models on page 472](#)
- Stress, see [Stress on page 1002](#)
- Trap space factor, see [Chapter 10 on page 349](#)
- Trap capture and emission rates, see [Local Capture and Emission Rates from PMI on page 357](#)
- Trap energy shift, see [Trap Energy Shift on page 1011](#)

34: Physical Model Interface

C++ Interface

- Piezoelectric polarization, see [Dependency of Saturation Velocity on Stress on page 633](#)
- Incomplete ionization, see [Chapter 6 on page 245](#)
- Hot-carrier injection, see [Chapter 18 on page 531](#)
- Piezoresistive coefficients, see [Piezoresistance Mobility Model on page 606](#)
- Spatial distribution function, see [Heavy Ions on page 492](#)

A separate interface is provided to add new entries to the current plot file, see [Current Plot File of Sentaurus Device on page 1027](#).

An interface is available that allows postprocessing of data during a transient simulation (see [Postprocess for Transient Simulation on page 1031](#)).

The following steps are needed to use a PMI model in a Sentaurus Device simulation:

- A C++ subroutine must be implemented to evaluate the PMI model. Additional C++ subroutines must be written to evaluate the derivatives of the PMI model with respect to all input variables (see [C++ Interface](#)).
- The `cmi` script produces a shared object file that Sentaurus Device loads at run-time (see [Shared Object Code on page 916](#)).
- The `PMIPath` variable must be defined in the `File` section of the command file. This defines the search path for the shared object files. A PMI model is activated in the `Physics` section of the command file by specifying its name (see [Command File of Sentaurus Device on page 916](#)).
- Parameters for PMI models can appear in the parameter file (see [Parameter File of Sentaurus Device on page 932](#)).

These steps are discussed further in the following sections. The source code for the examples is in the directory `$STROOT/tcad/$STRELEASE/lib/sdevice/src`.

C++ Interface

For each PMI model, you must implement a C++ subroutine to evaluate the model. Additional subroutines are necessary to evaluate the derivatives of the model with respect to all the input variables. More specifically, you must implement a C++ class that is derived from a base class declared in the header file `PMIModels.h`. In addition, a so-called virtual constructor function must be provided, which allocates an instance of the derived class.

For example, consider the implementation of Auger recombination as a new PMI model. (The built-in Auger recombination model is discussed in [Auger Recombination on page 322](#).)

In its simplest form, Auger recombination can be written as:

$$R_{\text{net}} = C \cdot (n + p) \cdot (np - n_{i,\text{eff}}^2) \quad (965)$$

Sentaurus Device needs to evaluate the value of R_{net} and the derivatives:

$$\begin{aligned} \frac{\partial R_{\text{net}}}{\partial n} &= C(np - n_{i,\text{eff}}^2 + (n + p)p) \\ \frac{\partial R_{\text{net}}}{\partial p} &= C(np - n_{i,\text{eff}}^2 + (n + p)n) \\ \frac{\partial R_{\text{net}}}{\partial n_{i,\text{eff}}} &= -2C(n + p)n_{i,\text{eff}} \end{aligned} \quad (966)$$

In the header file `PMIModels.h`, the following base class is defined for recombination models:

```
class PMI_Recombination : public PMI_Vertex_Interface {

public:
    PMI_Recombination (const PMI_Environment& env);
    virtual ~PMI_Recombination () ;

    virtual void Compute_r
        (const double t, const double n, const double p,
         const double nie, const double f, double& r) = 0;

    virtual void Compute_drdt
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdt) = 0;

    virtual void Compute_drdn
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdn) = 0;

    virtual void Compute_drdp
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdp) = 0;

    virtual void Compute_drdnie
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdn) = 0;

    virtual void Compute_drdf
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdf) = 0;
};
```

34: Physical Model Interface

C++ Interface

To implement a PMI model for Auger recombination, you must declare a derived class:

```
#include "PMIModels.h"

class Auger_Recombination : public PMI_Recombination {

    double C;

public:
    Auger_Recombination (const PMI_Environment& env);
    ~Auger_Recombination () ;

    void Compute_r
        (const double t, const double n, const double p,
         const double nie, const double f, double& r);

    void Compute_drdt
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdt);

    void Compute_drdn
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdn);

    void Compute_drdp
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdp);

    void Compute_drdnie
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdnie);

    void Compute_drdf
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdf);
};

};
```

The constructor of the derived class is invoked for each region of the device. In this example, the variable C is initialized from the parameter file:

```
Auger_Recombination::
Auger_Recombination (const PMI_Environment& env) :
    PMI_Recombination (env)
{ C = InitParameter ("C", 1e-30);
}
```

If the parameter C is not found in the parameter file, a default value of 10^{-30} is used (see [Parameter File of Sentaurus Device on page 932](#)). During a Newton iteration, Sentaurus Device evaluates a PMI model for each mesh vertex. The method `Compute_r()` computes the recombination rate for a given vertex. According to the parameter list, the recombination rate can depend on the following variables:

t	Lattice temperature
n	Electron density
p	Hole density
nie	Effective intrinsic density
f	Absolute value of electric field

The result of the function is stored in the parameter r:

```
void Auger_Recombination::
Compute_r (const double t, const double n, const double p,
          const double nie, const double f, double& r)
{ r = C * (n + p) * (n*p - nie*nie);
  if (r < 0.0) {
    r = 0.0;
  }
}
```

Besides `Compute_r()`, you must implement other methods to compute the partial derivatives of the recombination rate with respect to the input variables t, n, p, nie, and f. The implementation of `Compute_drdn()` to compute the value of $\partial R / \partial n$ is:

```
void Auger_Recombination::
Compute_drdn (const double t, const double n, const double p,
              const double nie, const double f, double& drdn)
{ double r = C * (n + p) * (n*p - nie*nie);
  if (r < 0.0) {
    drdn = 0.0;
  } else {
    drdn = C * ((n*p - nie*nie) + (n + p) * p);
  }
}
```

34: Physical Model Interface

Shared Object Code

Finally, you must provide a so-called virtual constructor function, which allocates a variable of the new class:

```
extern "C"  
PMI_Recombination* new_PMI_Recombination (const PMI_Environment& env)  
{ return new Auger_Recombination (env);  
}
```

NOTE This function must have C linkage and exactly the same name as declared in the header file `PMIModels.h`.

Shared Object Code

Sentaurus Device assumes that the shared object code corresponding to a PMI model can be found in the file `modelname.so.arch`. The base name of this file must be identical to the name of the PMI model. The extension `.arch` depends on the hardware architecture. The script `cmi`, which is also a part of the CMI, can be used to produce the shared object files (see [Compact Models User Guide, Run-Time Support on page 133](#)).

Command File of Sentaurus Device

To load PMI models into Sentaurus Device, the `PMIPath` search path must be defined in the `File` section of the command file. The value of `PMIPath` consists of a sequence of directories, for example:

```
File {  
    PMIPath = ". /home/joe/lib /home/mary/sdevice/lib"  
}
```

For each PMI model, which appears in the `Physics` section, the given directories are searched for a corresponding shared object file `modelname.so.arch`.

The PMI in Sentaurus Device provides access to mesh-based scalar fields specified by you. These fields must be defined on the device grid in a separate DF–ISE (extension `.dat`) or TDR (extension `.tdr`) data file. Up to ten datasets (`PMIUserField0`, ..., `PMIUserField9`) can be defined. Sentaurus Device reads the user-defined fields if the corresponding file name is given in the command file:

```
File {  
    PMIUserFields = "fields"  
}
```

A PMI model can be activated in the `Physics` section of the command file by specifying the name of the PMI model in the appropriate part of the `Physics` section. Examples for different types of PMI models are:

- Generation–recombination models:

```
Physics {  
    Recombination (pmi_model_name ...)  
}
```

- Avalanche generation:

```
Physics {  
    Recombination (Avalanche (pmi_model_name ...))  
}
```

- Mobility models:

```
Physics {  
    Mobility {  
        DopingDependence (pmi_model_name)  
        Enormal (pmi_model_name)  
        ToCurrentEnormal (pmi_model_name)  
        HighFieldSaturation (pmi_model_name driving_force)  
    }  
}
```

A PMI model name can only consist of alphanumeric characters and underscores (`_`). The first character must be either a letter or an underscore. A PMI model name can also be quoted as "`"model_name"`" to avoid conflicts with Sentaurus Device keywords.

All the PMI models can be specified regionwise or materialwise:

```
Physics (region = "Region.1") {  
    ...  
}  
Physics (material = "AlGaAs") {  
    ...  
}
```

Certain values of PMI models can be plotted in the `Plot` section of the command file. The following identifiers are recognized:

- Generation–recombination models:

```
Plot {  
    PMIRecombination  
}
```

- User-defined fields:

```
Plot {
    PMIUserField0 PMIUserField1 PMIUserField2 PMIUserField3
    PMIUserField4 PMIUserField5 PMIUserField6 PMIUserField7
    PMIUserField8 PMIUserField9
}
```

- Piezoelectric polarization:

```
Plot {
    PE_Polarization/vector PE_Charge
}
```

The current plot PMI can be used to add entries to the current plot file:

```
CurrentPlot {
    pmi_CurrentPlot
}
```

Vertex-based Run-Time Support

The base class `PMI_Vertex_Interface` provides run-time support for vertex-based PMI models:

```
class PMI_Vertex_Interface {
public:
    PMI_Vertex_Interface (const PMI_Environment& env);
    virtual ~PMI_Vertex_Interface ();

    const char* Name () const;

    const char* ReadRegionName () const;

    const char* ReadRegionMaterial () const;

    const char* ReadDeviceName () const;

    const PMIBaseParam* ReadParameter (const char* name) const;

    double InitParameter (const char* name, double defaultvalue) const;

    int ReadDimension () const;

    void ReadCoordinate (double& x, double& y, double& z) const;

    double ReadDistanceFromSemiconductorInsulatorInterface() const;
```

```

double ReadDistanceFromHighkInsulator() const;

double ReadTime () const;

double ReadTransientStepSize() const;

int ReadTransientStepType() const;

double ReadxMoleFraction () const;
double ReadyMoleFraction () const;

double ReadDoping (PMI_DopingSpecies species) const;

double ReadDoping (const char* SpeciesName) const;

int IsUserFieldDefined (PMI_UserFieldIndex index) const;

double ReadUserField (PMI_UserFieldIndex index) const;

void WriteUserField (PMI_UserFieldIndex index, double value);

double ReadStress (PMI_StressIndex index) const;

int ReadNumberOfThreads () const;
int ReadThreadId () const;
};

```

The method `Name()` returns the name of the PMI model as specified in the command file. Similarly, `ReadRegionName()`, `ReadRegionMaterial()`, and `ReadDeviceName()` return the name of the current region, the material of the current region, and the name of the device, respectively. The methods `ReadParameter()` and `InitParameter()` read the value of a parameter from the parameter file (see [Parameter File of Sentaurus Device on page 932](#)).

`ReadDimension()` returns the dimension of the problem. The function `ReadCoordinate()` provides the coordinates of the current vertex [μm].

The functions `ReadTime()` and `ReadTransientStepSize()` return the simulation time and the current step size during a transient simulation [s]. `ReadTransientStepType()` provides access to the actual transient step type. The enumeration type `PMI_StepType` is defined for identification:

```

enum PMI_StepType {
    PMI_UndefinedStepType = 0,
    PMI_TR = 1,
    PMI_BDF = 2,
    PMI_BE = 3
}

```

34: Physical Model Interface

Vertex-based Run-Time Support

`ReadDistanceFromSemiconductorInsulatorInterface()` returns the distance of the current vertex to the nearest semiconductor-insulator interface (in μm) if the current vertex is in a semiconductor region; otherwise, the function returns minus that distance.

`ReadDistanceFromHighkInsulator()` returns the distance of the current vertex to the nearest high-k insulator (in μm). If no high-k insulator is found in the structure, the function returns a value < 0 .

The methods `ReadxMoleFraction()` and `ReadyMoleFraction()` return the x and y mole fractions, respectively.

The methods `ReadDoping(species)` and `ReadDoping(SpeciesName)` return the doping profiles for the current vertex [cm^{-3}]. The string `SpeciesName` is the same as in the file `datexcodes.txt` (see [User-Defined Species on page 125](#)). The enumeration type `PMI_DopingSpecies` is used to select the doping species, the incomplete ionization doping species, and their derivatives:

```
enum PMI_DopingSpecies {
    // Acceptors
    PMI_BoronActive,           // active Boron concentration
    PMI_BoronChemical,         // chemical Boron concentration
    PMI_IndiumActive,          // active Indium concentration
    PMI_IndiumChemical,        // chemical Indium concentration
    PMI_PDopantActive,         // active PDopant concentration
    PMI_PDopantChemical,       // chemical PDopant concentration
    PMI_Acceptor,              // total acceptor concentration

    // incomplete ionization entries
    PMI_AcceptorMinus,         // total incomplete ionization acceptor concentration
    PMI_AcceptorMinusPer_hDensity,
    PMI_AcceptorMinusPerT,

    // Donors
    PMI_PhosphorusActive,      // active Phosphorus concentration
    PMI_PhosphorusChemical,    // chemical Phosphorus concentration
    PMI_ArsenicActive,         // active Arsenic concentration
    PMI_ArsenicChemical,       // chemical Arsenic concentration
    PMI_AntimonyActive,        // active Antimony concentration
    PMI_AntimonyChemical,      // chemical Antimony concentration
    PMI_NitrogenActive,         // active Nitrogen concentration
    PMI_NitrogenChemical,       // chemical Nitrogen concentration
    PMI_NDopantActive,          // active NDopant concentration
    PMI_NDopantChemical,        // chemical NDopant concentration
    PMI_Donor,                 // total donor concentration

    // incomplete ionization entries
    PMI_DonorPlus,             // total incomplete ionization donor concentration
}
```

```
    PMI_DonorPlusPer_eDensity,
    PMI_DonorPlusPerT
};
```

NOTE The species `PMI_Acceptor` and `PMI_Donor` are always defined. The remaining entries are only defined if they appear in the doping input file. The incomplete ionization entries are only accessible if the option `IncompleteIonization` is activated (see [Chapter 6 on page 245](#)).

The method `IsUserFieldDefined()` checks if a user-defined field has been specified. The enumeration type `PMI_UserFieldIndex` selects the desired field:

```
enum PMI_UserFieldIndex {
    PMI_UserField0, PMI_UserField1, PMI_UserField2, PMI_UserField3,
    PMI_UserField4, PMI_UserField5, PMI_UserField6, PMI_UserField7,
    PMI_UserField8, PMI_UserField9
};
```

If a specific user-field is defined, the method `ReadUserField()` reads its value for the current vertex. For time-dependent user-fields, this is the value written in the last successful time step. `WriteUserField()` allows the modification of the value of a specific user-field for the current vertex. It writes to an auxiliary field. After a transient time step is finished, this auxiliary field becomes the field accessible with `ReadUserField()`.

The method `ReadStress()` returns the value of one of the following stress components:

```
enum PMI_StressIndex {
    PMI_StressXX, PMI_StressYY, PMI_StressZZ,
    PMI_StressYZ, PMI_StressXZ, PMI_StressXY
};
```

The two methods `ReadNumberOfThreads()` and `ReadThreadId()` are useful for writing thread-safe PMI code that can be invoked during the parallel assembly. `ReadThreadId()` returns the thread identifier of the calling thread. The result is a number from 0 to $n - 1$, where n is the total number of threads as returned by `ReadNumberOfThreads()`.

Vertex-based Run-Time Support for Multistate Configuration-dependent Models

The base class `PMI_MSC_Vertex_Interface` provides the same support as `PMI_Vertex_Interface`, plus additional functions needed by models that depend on a multistate configuration (see [Chapter 11 on page 365](#)):

```
class PMI_MSC_Vertex_Interface : public PMI_Vertex_Interface
{
public:
    PMI_MSC_Vertex_Interface(const PMI_Environment&,
        const std::string& msconfig_name,
        int model_index,
        const std::string& model_string);
public:
    const std::string& msconfig_name () const;
    size_t nb_states () const;
    std::string& state (size_t index) const;
    int model_index () const;
    const std::string& model_string () const;
    virtual void init_parameter (){};
};
```

The constructor argument `msconfig_name` determines the name of the multistate configuration on which the model depends, and the constructor arguments `model_index` and `model_string` are an integer and a string that you can evaluate in your model. You call the constructor `PMI_MSC_Vertex_Interface` only indirectly using constructors of base classes of multistate configuration-dependent PMIs.

The function `nb_states` returns the number of states in the selected multistate configuration, `state` returns the name of a particular state, and `msconfig_name`, `model_index`, and `model_string` return the arguments of the constructor of the same name.

The function `init_parameter` is always called before the parameters are changed. It allows you to keep model-internal data up-to-date in cases such as ramping of parameters.

Mesh-based Run-Time Support

The base class `PMI_Device_Interface` provides run-time support for mesh-based PMI models:

```
class PMI_Device_Interface {  
  
public:  
    PMI_Device_Interface (const PMI_Device_Environment& env);  
    virtual ~PMI_Device_Interface ();  
  
    const char* Name () const;  
  
    const PMIBaseParam* ReadParameter (const char* name) const;  
    double InitParameter (const char* name, double defaultvalue) const;  
  
    double ReadTime () const;  
  
    double ReadTransientStepSize () const;  
  
    int ReadTransientStepType () const;  
  
    const des_mesh* Mesh () const;  
    des_data* Data () const;  
};
```

The method `Name()` returns the name of the PMI model as specified in the command file. The methods `ReadParameter()` and `InitParameter()` read the value of a parameter from the parameter file (see [Parameter File of Sentaurus Device on page 932](#)).

NOTE Parameters for a mesh-based PMI must appear in the global parameter section. Regionwise or materialwise parameters are not supported.

The functions `ReadTime()` and `ReadTransientStepSize()` return the simulation time and the current step size during a transient simulation [s]. `ReadTransientStepType()` provides access to the actual transient step type. The enumeration type `PMI_StepType` is defined for identification:

```
enum PMI_StepType {  
    PMI_UndefinedStepType = 0,  
    PMI_TR = 1,  
    PMI_BDF = 2,  
    PMI_BE = 3  
}
```

The methods `Mesh()` and `Data()` provide access to the mesh and data of Sentaurus Device (see [Device Mesh](#) and [Device Data on page 930](#)).

Device Mesh

A device mesh of Sentaurus Device consists of a number of regions. A region is either a contact region consisting of a list of contact vertices or a bulk region consisting of a list of elements. An element is described by a list of vertices.

Vertex

In the file `PMIModels.h`, the class `des_vertex` is declared as follows:

```
class des_vertex {  
  
public:  
    size_t index () const;  
  
    const double* coord () const;  
  
    bool equal_coord (des_vertex* v) const;  
  
    size_t size_edge () const;  
    des_edge* edge (size_t i) const;  
  
    size_t size_element () const;  
    des_element* element (size_t i) const;  
  
    size_t size_region () const;  
    des_region* region (size_t i) const;  
  
    size_t size_regioninterface () const;  
    des_regioninterface* regioninterface (size_t i) const;  
};
```

The value of `index()` can be used as an index for vertex-based data (see [Device Data on page 930](#)).

The location of a vertex [μm] is given by its coordinates `coord()`. The function `equal_coord()` should be used to check if two vertices have the same coordinates. For example, Sentaurus Device duplicates vertices along heterointerfaces. Consequently, two vertices with different indices can share the same coordinates.

`size_edge()` returns the number of edges connected to a vertex. The method `edge()` can be used to retrieve the i -th edge.

`size_region()` returns the number of regions containing a vertex. The method `region()` can be used to retrieve the i-th region.

`size_regioninterface()` reports how many region interfaces a vertex belongs to. The i-th region interface is returned by the method `regioninterface()`.

Edge

In the file `PMIModels.h`, the class `des_edge` is declared as follows:

```
class des_edge {  
  
public:  
    size_t index () const;  
  
    des_vertex* start () const;  
    des_vertex* end () const;  
  
    size_t size_element () const;  
    des_element* element (size_t i) const;  
  
    size_t size_region () const;  
    des_region* region (size_t i) const;  
};
```

The value of `index()` can be used as an index for edge-based data (see [Device Data on page 930](#)).

`start()` and `end()` return the first and second vertex connected to the edge, respectively.

`size_element()` returns the number of elements connected to an edge. The method `element()` can be used to retrieve the i-th element.

`size_region()` returns the number of regions containing an edge. The method `region()` can be used to retrieve the i-th region.

Element

In the file `PMIModels.h`, the class `des_element` is declared as follows:

```
class des_element {  
  
public:  
    typedef enum { point, line, triangle, rectangle, tetrahedron,  
                  pyramid, prism, cuboid, tetrabrick } des_type;
```

34: Physical Model Interface
Mesh-based Run-Time Support

```

size_t index () const;

des_type type () const;

size_t size_vertex () const;
des_vertex* vertex (size_t i) const;

size_t size_edge () const;
des_edge* edge (size_t i) const;

des_bulk* bulk () const;
};

```

Figure 120 shows the numbering of vertices and edges for all element types.

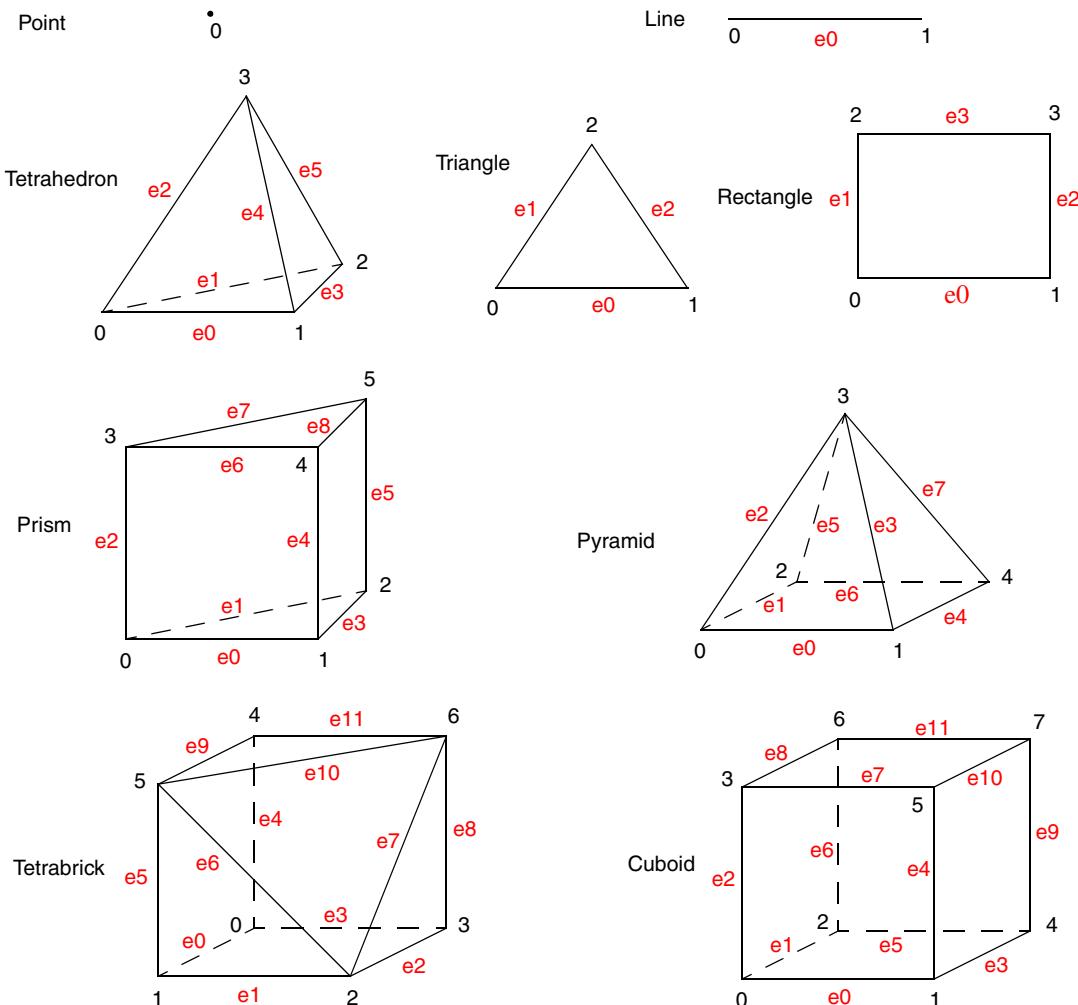


Figure 120 Vertex and edge numbering

The value of `index()` can be used as an index for element-based data (see [Device Data on page 930](#)).

`type()` returns the type of an element (point, line, triangle, rectangle, tetrahedron, pyramid, prism, cuboid, or tetrabrick). An element is mainly described by its vertices. `size_vertex()` returns the number of vertices in an element, and the method `vertex()` can be used to retrieve the *i*-th vertex. `size_edge()` returns the number of edges of an element. The method `edge()` can be used to retrieve the *i*-th edge. The method `bulk()` returns the bulk region containing the element.

Region

In the file `PMIModels.h`, the base class `des_region` is declared as follows:

```
class des_region {

public:
    typedef enum { bulk, contact } des_type;

    virtual des_type type () const = 0;

    std::string name () const;

    size_t size_vertex () const;
    des_vertex* vertex (size_t i) const;

    size_t size_edge () const;
    des_edge* edge (size_t i) const;
};
```

A mesh of Sentaurus Device consists of two types of regions: bulk regions and contacts. The virtual method `type()` returns the type of a region. The name of a region is returned by `name()`. `size_vertex()` returns the number of vertices in a region. The method `vertex()` can be used to retrieve the *i*-th vertex. `size_edge()` returns the number of edges in a region. The method `edge()` can be used to retrieve the *i*-th edge.

The class `des_bulk` is derived from `des_region`:

```
class des_bulk : public des_region {

public:
    des_type type () const;

    std::string material () const;

    size_t size_element () const;
    des_element* element (size_t i) const;
```

```
    size_t size_regioninterface () const;
    des_regioninterface* regioninterface (size_t i) const;
};
```

`material()` returns the name of the material in a bulk region. `size_element()` returns the number of elements in a region. The method `element()` can be used to retrieve the *i*-th element. `size_regioninterface()` reports how many region interfaces are connected to this bulk region. The *i*-th region interface can then be retrieved by the method `regioninterface()`.

Similarly, the class `des_contact` is also derived from `des_region`:

```
class des_contact : public des_region {

public:
    des_type type () const;
};
```

Region Interface

A region interface separates two bulk regions. It is described by the following class:

```
class des_regioninterface {

public:
    size_t index () const;

    des_bulk* bulk1 () const;
    des_bulk* bulk2 () const;

    bool is_heterointerface () const;

    size_t size_vertex () const;
    des_vertex* vertex (size_t i) const;
    size_t index (size_t local_vertex_index) const;
};
```

The value of `index()` is used to access the surface measure array (see [Device Data on page 930](#)).

The two bulk regions connected to a region interface are returned by `bulk1()` and `bulk2()`.

Use `is_heterointerface()` to determine if double points exist for this interface. For a heterointerface, each vertex belongs to either region 1 or region 2, but not both. For a regular interface, each vertex belongs to both region 1 and region 2.

The number of vertices contained in a region interface is returned by the method `size_vertex()`. The *i*-th vertex can be obtained by invoking `vertex()`. The method `index(size_t local_vertex_index)` is used to obtain the correct index for interface-based data (see [Device Data on page 930](#)).

Mesh

In the file `PMIModels.h`, the class `des_mesh` is declared as follows:

```
class des_mesh {  
  
public:  
    int dim () const;  
  
    size_t size_vertex () const;  
    des_vertex* vertex (size_t i) const;  
  
    size_t size_edge () const;  
    des_edge* edge (size_t i) const;  
  
    size_t size_element () const;  
    des_element* element (size_t i) const;  
  
    size_t size_region () const;  
    des_region* region (size_t i) const;  
  
    size_t size_regioninterface () const;  
    des_regioninterface* regioninterface (size_t i) const;  
};
```

The dimension of the mesh is given by `dim()`. The possible values are 1, 2, and 3.

`size_vertex()` returns the number of vertices in the mesh. The method `vertex()` can be used to retrieve the *i*-th vertex. `size_edge()` returns the number of edges in the mesh. The method `edge()` can be used to retrieve the *i*-th edge. `size_element()` returns the number of elements in the mesh. The method `element()` can be used to retrieve the *i*-th element. `size_region()` returns the number of regions in the mesh. The method `region()` can be used to retrieve the *i*-th region. There are `size_regioninterface()` region interfaces in the mesh, and the *i*-th interface is returned by invoking `regioninterface()`.

Device Data

In the file `PMIModels.h`, the class `des_data` is declared as follows:

```
class des_data {  
  
public:  
    typedef enum { vertex, edge, element, rivertex } des_location;  
  
    const double*const* ReadCoefficient ();  
    const double*const* ReadMeasure ();  
    const double*const* ReadSurfaceMeasure ();  
  
    const double* ReadScalar (des_location location, std::string name);  
    const double*const* ReadVector (des_location location, std::string name);  
    void WriteScalar (des_location location, std::string name, const double*  
        newvalue);  
  
    const double* const* ReadGradient(des_location location, std::string name);  
  
    const double* ReadFlux(des_location location, std::string name);  
};
```

The methods `ReadCoefficient()` and `ReadMeasure()` return the box method coefficients κ_{ij} and measure μ_{ij} used in Sentaurus Device (see [Discretization on page 885](#)).

`ReadCoefficient()` returns a two-dimensional array. The two indices are the element index and the local edge number. The units are μm^{-1} in 1D, 1 in 2D, and μm in 3D.

The following code fragment reads the coefficients for all element edges:

```
const des_mesh* mesh = Mesh();  
des_data* data = Data();  
const double*const* coeff = data->ReadCoefficient();  
for (size_t eli = 0; eli < mesh->size_element(); eli++) {  
    des_element* el = mesh->element(eli);  
    for (size_t ei = 0; ei < el->size_edge(); ei++) {  
        des_edge* e = el->edge(ei);  
        const double c = coeff[el->index()][ei];  
    }  
}
```

NOTE The values κ_{ij} returned by `ReadCoefficient()` are element-edge coefficients. The edge coefficients κ_i can be obtained by adding the contributions from all elements connected to an edge i .

`ReadMeasure()` returns a two-dimensional array. The two indices are the element index and the local vertex number. The units are μm in 1D, μm^2 in 2D, and μm^3 in 3D.

The following code fragment reads the measures for all element vertices:

```
const des_mesh* mesh = Mesh();
des_data* data = Data();
const double*const* measure = data->ReadMeasure();
for (size_t eli = 0; eli < mesh->size_element(); eli++) {
    des_element* el = mesh->element(eli);
    for (size_t vi = 0; vi < el->size_vertex(); vi++) {
        des_vertex* v = el->vertex(vi);
        const double m = measure[el->index()][vi];
    }
}
```

NOTE The values μ_{ij} returned by `ReadMeasure()` are element–vertex measures. The node measures μ_i can be obtained by adding the contributions from all elements connected to a vertex i .

The method `ReadSurfaceMeasure()` provides the surface measure associated with region interface vertices (edge length [μm] in 2D and surface area [μm^2] in 3D). The two indices are the region interface index and the local vertex number.

The methods `ReadScalar()`, `ReadVector()`, and `WriteScalar()` provide access to the data of Sentaurus Device. The values can be located on vertices, elements, edges, or region interfaces. See [Table 121 on page 1060](#) and [Table 122 on page 1084](#) for an overview of available scalar and vector data.

`ReadScalar()` returns a read-only one-dimensional array. Use the `index()` method in the classes `des_vertex`, `des_edge`, or `des_element` to access the array elements. The proper addressing of region interface datasets is shown in the code fragment below.

`ReadVector()` returns a read-only two-dimensional array. The first index selects the dimension (0, 1, 2) and the second index is used in the same way as for scalar data.

`WriteScalar()` writes back a one-dimensional array. The organization of the array is the same as for the arrays obtained with `ReadScalar()`. You must ensure that the size of the array is sufficient to hold all required entries.

`ReadGradient()` returns a 2D array that contains, for each vertex, the gradient of a chosen variable. Thereby, the first index selects the partial derivative:

$$[0] = \left(\frac{\partial}{\partial x}\right), [1] = \left(\frac{\partial}{\partial y}\right), [2] = \left(\frac{\partial}{\partial z}\right) \quad (967)$$

The second index identifies the vertex. The actual implementation of `ReadGradient()` works for vertex-based datasets only.

`ReadFlux()` returns an array that contains, for each vertex, the surface integral of the gradient of a chosen variable taken over the boundary of the box divided by the box volume. The actual implementation of `ReadFlux()` works for vertex-based datasets only.

The following code fragment traverses all region interfaces and reads the surface measure and a dataset for each region interface vertex:

```
const des_mesh* mesh = Mesh();
des_data* data = Data();
const double*const* surface = data->ReadSurfaceMeasure();
const double* hot_elec = data->ReadScalar(des_data::rivertex,
"HotElectronInj");
for (size_t rii = 0; rii < mesh->size_regioninterface(); rii++) {
    des_regioninterface* ri = mesh->regioninterface(rii);
    for (size_t vi = 0; vi < ri->size_vertex(); vi++) {
        des_vertex* v = ri->vertex(vi);
        const double sm = surface[ri->index()][vi];
        const double he = hot_elec[ri->index(vi)];
    }
}
```

Parameter File of Sentaurus Device

For each PMI model, a corresponding section with the same name can appear in the parameter file:

```
PMI_model_name {
    par1 = value
    par2 = value
    ...
}
```

NOTE Parameter names can only consist of alphanumeric characters and underscores (_). The first character must be either a letter or an underscore.

The parameters can be specified regionwise and materialwise:

```
Region = "Region.1" {
    PMI_model_name {
        ...
    }
}
```

```
Material = "AlGaAs" {
    PMI_model_name {
        ...
    }
}
```

The method `PMI_Vertex_Interface::ReadParameter()` can be used to obtain the value of a parameter given its name. `ReadParameter()` returns a pointer to a variable of type `PMIBaseParam`. A NULL pointer indicates that the parameter has not been defined in the parameter file. If the pointer `p` in:

```
const PMIBaseParam* p = ReadParameter ("name of parameter");
```

is not NULL, the following assignment is a valid C++ statement:

```
double d = *p;
```

The variable `d` is assigned the value of the parameter `p`. The method `PMI_Vertex_Interface::InitParameter()` checks if a parameter has been specified in the parameter file. If the parameter has been specified, the given value is taken. Otherwise, the default value is used.

Generation–Recombination Model

The recombination rate R_{net} appears in the electron and hole continuity equations (see [Eq. 27](#), [p. 178](#)).

Dependencies

The recombination rate R_{net} may depend on these variables:

<code>t</code>	Lattice temperature [K]
<code>n</code>	Electron density [cm^{-3}]
<code>p</code>	Hole density [cm^{-3}]
<code>nie</code>	Effective intrinsic density [cm^{-3}]
<code>f</code>	Absolute value of electric field [Vcm^{-1}]

The PMI model must compute the following results:

r	Generation–recombination rate [$\text{cm}^{-3}\text{s}^{-1}$]
drdt	Derivative of r with respect to t [$\text{cm}^{-3}\text{s}^{-1}\text{K}^{-1}$]
drdn	Derivative of r with respect to n [s^{-1}]
drdp	Derivative of r with respect to p [s^{-1}]
drdnie	Derivative of r with respect to nie [s^{-1}]
drdf	Derivative of r with respect to f [$\text{cm}^{-2}\text{s}^{-1}\text{V}^{-1}$]

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Recombination : public PMI_Vertex_Interface {

public:
    PMI_Recombination (const PMI_Environment& env);
    virtual ~PMI_Recombination () ;

    virtual void Compute_r
        (const double t, const double n, const double p,
         const double nie, const double f, double& r) = 0;

    virtual void Compute_drdt
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdt) = 0;

    virtual void Compute_drdn
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdn) = 0;

    virtual void Compute_drdp
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdp) = 0;

    virtual void Compute_drdnie
        (const double t, const double n, const double p,
         const double nie, const double f, double& drdnie) = 0;
```

```
virtual void Compute_drdf
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdf) = 0;
};
```

The prototype for the virtual constructor is:

```
typedef PMI_Recombination* new_PMI_Recombination_func
    (const PMI_Environment& env);
extern "C" new_PMI_Recombination_func new_PMI_Recombination;
```

By default, Sentaurus Device assumes that a PMI generation–recombination model depends on the electric field. However, you can implement the optional function `PMI_Recombination_ElectricField()` to indicate whether the model depends on the electric field. If the model does not depend on the electric field (return value of 0), the method `Compute_drdf()` is not called, and the matrix assembly in Sentaurus Device works more efficiently:

```
typedef int PMI_Recombination_ElectricField_func ();
extern "C"
PMI_Recombination_ElectricField_func PMI_Recombination_ElectricField;
```

Example: Auger Recombination

See [C++ Interface on page 912](#).

Avalanche Generation Model

The generation rate due to impact ionization can be expressed as:

$$G^{\parallel} = \alpha_n n v_n + \alpha_p p v_p \quad (968)$$

where α_n and α_p are the ionization coefficients for electrons and holes, respectively (compare with [Eq. 307, p. 321](#)). The PMI in Sentaurus Device allows you to redefine the calculation of α_n and α_p .

Dependencies

The ionization coefficients α_n and α_p may depend on the following variables:

F	Driving force [Vcm ⁻¹]
t	Lattice temperature [K]
bg	Band gap [eV]
ct	Carrier temperature [K]
currentWoMob [3]	Current without mobility [cm ⁻⁴ AVs]

The parameter `ct` represents the electron temperature during the calculation of α_n and the hole temperature during the calculation of α_p .

The parameter `currentWoMob` can be used to compute anisotropic avalanche generation. Only the first d components of the vector `currentWoMob` are defined, where d is equal to the dimension of the problem. It is recommended that only the direction of the vector `currentWoMob` is taken into account, but not its magnitude.

The PMI model must compute the following results:

alpha	Ionization coefficient [cm ⁻¹]
dalphadF	Derivative of alpha with respect to F [V ⁻¹]
dalphadt	Derivative of alpha with respect to t [cm ⁻¹ K ⁻¹]
dalphadbg	Derivative of alpha with respect to bg [cm ⁻¹ eV ⁻¹]
dalphadct	Derivative of alpha with respect to ct [cm ⁻¹ K ⁻¹]
dalphadcurrentWoMob [3]	Derivative of alpha with respect to currentWoMob [cm ³ A ⁻¹ V ⁻¹ s ⁻¹]

Only the first d components of the vector `dalphadcurrentWoMob` need to be computed.

C++ Interface

Different driving forces for avalanche generation can be selected in the command file. The enumeration type `PMI_AvalancheDrivingForce`, defined in `PMIModels.h`, is used to reflect your selection:

```
enum PMI_AvalancheDrivingForce {
    PMI_AvalancheElectricField,
    PMI_AvalancheParallelElectricField,
    PMI_AvalancheGradQuasiFermi,
    PMI_AvalancheCarrierTemperatureCanali
};
```

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Avalanche : public PMI_Vertex_Interface {

private:
    const PMI_AvalancheDrivingForce drivingForce;

public:
    PMI_Avalanche (const PMI_Environment& env,
                   const PMI_AvalancheDrivingForce force);
    virtual ~PMI_Avalanche () ;

    PMI_AvalancheDrivingForce AvalancheDrivingForce () const
    { return drivingForce; }

    virtual void Compute_alpha
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& alpha) = 0;

    virtual void Compute_dalphadF
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadF) = 0;

    virtual void Compute_dalphadt
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadt) = 0;

    virtual void Compute_dalphadb
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadb) = 0;

    virtual void Compute_dalphadct
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadct) = 0;
```

34: Physical Model Interface

Avalanche Generation Model

```
virtual void Compute_dalphadcurrentWoMob
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double
     dalphadcurrentWoMob[3]) = 0;
};
```

Two virtual constructors are required for the calculation of the ionization coefficients α_n and α_p :

```
typedef PMI_Avalanche* new_PMI_Avalanche_func
(const PMI_Environment& env, const PMI_AvalancheDrivingForce force);
extern "C" new_PMI_Avalanche_func new_PMI_e_Avalanche;
extern "C" new_PMI_Avalanche_func new_PMI_h_Avalanche;
```

Example: Okuto Model

Okuto and Crowell propose the following expression for the ionization coefficient α :

$$\alpha(F) = a[1 + c(T - T_0)]F \exp\left[-\left(\frac{b[1 + d(T - T_0)]}{F}\right)^2\right] \quad (969)$$

This built-in model is discussed in [Okuto–Crowell Model on page 326](#) and its implementation as a PMI model is:

```
#include "PMIModels.h"

class Okuto_Avalanche : public PMI_Avalanche {

protected:
    const double T0;
    double a, b, c, d;

public:
    Okuto_Avalanche (const PMI_Environment& env,
                      const PMI_AvalancheDrivingForce force);

    ~Okuto_Avalanche () ;

    void Compute_alpha
        (const double F, const double t, const double bg,
         const double ct, const double currentWoMob[3], double& alpha);

    void Compute_dalphadF
        (const double F, const double t, const double bg,
         const double ct, const double currentWoMob[3], double& dalphadF);
```

```

void Compute_dalphadt
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadt);

void Compute_dalphadbg
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadbg);

void Compute_dalphadct
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadct);

void Compute_dalphadcurrentWoMob
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double
dalphadcurrentWoMob[3]);
};

Okuto_Avalanche::
Okuto_Avalanche (const PMI_Environment& env,
                  const PMI_AvalancheDrivingForce force) :
    PMI_Avalanche (env, force),
    T0 (300.0)
{
}

Okuto_Avalanche::
~Okuto_Avalanche ()
{
}

void Okuto_Avalanche::
Compute_alpha (const double F, const double t, const double bg,
               const double ct, const double currentWoMob[3], double& alpha)
{ const double aa = a * (1.0 + c * (t - T0));
  const double bb = b * (1.0 + d * (t - T0)) / F;
  alpha = aa * F * exp (-bb*bb);
}

void Okuto_Avalanche::
Compute_dalphadF (const double F, const double t, const double bg,
                   const double ct, const double currentWoMob[3], double&
dalphadF)
{ const double aa = a * (1.0 + c * (t - T0));
  const double bb = b * (1.0 + d * (t - T0)) / F;
  const double alpha = aa * F * exp (-bb*bb);
  dalphadF = (alpha / F) * (1.0 + 2.0*bb*bb);
}

```

34: Physical Model Interface

Avalanche Generation Model

```
void Okuto_Avalanche::  
Compute_dalphadt (const double F, const double t, const double bg,  
                  const double ct, const double currentWoMob[3], double&  
                  dalphadt)  
{ const double aa = a * (1.0 + c * (t - T0));  
  const double bb = b * (1.0 + d * (t - T0)) / F;  
  const double tmp = F * exp (-bb*bb);  
  dalphadt = tmp * (a * c - 2.0 * aa * bb * b * d / F);  
}  
  
void Okuto_Avalanche::  
Compute_dalphadbg (const double F, const double t, const double bg,  
                   const double ct, const double currentWoMob[3], double&  
                   dalphadbg)  
{ dalphadbg = 0.0;  
}  
  
void Okuto_Avalanche::  
Compute_dalphadct (const double F, const double t, const double bg,  
                   const double ct, const double currentWoMob[3], double&  
                   dalphadct)  
{ dalphadct = 0.0;  
}  
  
void Okuto_Avalanche::  
Compute_dalphadcurrentWoMob (const double F, const double t, const double bg,  
                             const double ct, const double currentWoMob[3],  
                             double dalphadcurrentWoMob[3])  
{ const int dim = ReadDimension ();  
  for (int k = 0; k < dim; k++) {  
    dalphadcurrentWoMob [k] = 0.0;  
  }  
}  
  
class Okuto_e_Avalanche : public Okuto_Avalanche {  
  
public:  
  Okuto_e_Avalanche (const PMI_Environment& env,  
                     const PMI_AvalancheDrivingForce force);  
  
  ~Okuto_e_Avalanche () {}  
};  
  
Okuto_e_Avalanche::  
Okuto_e_Avalanche (const PMI_Environment& env,  
                  const PMI_AvalancheDrivingForce force) :  
  Okuto_Avalanche (env, force)
```

```
{ // default values
    a = InitParameter ("a_e", 0.426);
    b = InitParameter ("b_e", 4.81e5);
    c = InitParameter ("c_e", 3.05e-4);
    d = InitParameter ("d_e", 6.86e-4);
}

class Okuto_h_Avalanche : public Okuto_Avalanche {

public:
    Okuto_h_Avalanche (const PMI_Environment& env,
                        const PMI_AvalancheDrivingForce force);

    ~Okuto_h_Avalanche () {}

};

Okuto_h_Avalanche::Okuto_h_Avalanche (const PMI_Environment& env,
                                       const PMI_AvalancheDrivingForce force) :
    Okuto_Avalanche (env, force)
{ // default values
    a = InitParameter ("a_h", 0.243);
    b = InitParameter ("b_h", 6.53e+5);
    c = InitParameter ("c_h", 5.35e-4);
    d = InitParameter ("d_h", 5.67e-4);
}

extern "C"
PMI_Avalanche* new_PMI_e_Avalanche
    (const PMI_Environment& env, const PMI_AvalancheDrivingForce force)
{ return new Okuto_e_Avalanche (env, force);
}

extern "C"
PMI_Avalanche* new_PMI_h_Avalanche
    (const PMI_Environment& env, const PMI_AvalancheDrivingForce force)
{ return new Okuto_h_Avalanche (env, force);
}
```

Mobility Models

Sentaurus Device supports three types of PMI mobility models:

- Doping-dependent mobility
- Mobility degradation at interfaces
- High-field saturation

PMI and built-in models can be used simultaneously. See [Chapter 8 on page 267](#) for more information about how the contributions of the different models are combined. PMI mobility models support anisotropic calculations and can be evaluated along different crystallographic axes.

The enumeration type:

```
enum PMI_AnisotropyType {  
    PMI_Isotropic,  
    PMI_Anisotropic  
};
```

determines the axis. The default is isotropic mobility. If anisotropic mobilities are activated in the command file, the PMI mobility classes are also instantiated in the anisotropic direction.

Doping-dependent Mobility

A doping-dependent PMI model must account for both the constant mobility and doping-dependent mobility models discussed in [Mobility due to Phonon Scattering on page 268](#) and [Doping-dependent Mobility Degradation on page 268](#).

Dependencies

The constant mobility and doping-dependent mobility μ_{dop} may depend on the following variables:

- | | |
|---|---------------------------------------|
| t | Lattice temperature [K] |
| n | Electron density [cm^{-3}] |
| p | Hole density [cm^{-3}] |

The PMI model must compute the following results:

m	Mobility μ_{dop} [$\text{cm}^2 \text{V}^{-1} \text{s}^{-1}$]
dmdn	Derivative of μ_{dop} with respect to n [$\text{cm}^5 \text{V}^{-1} \text{s}^{-1}$]
dmdp	Derivative of μ_{dop} with respect to p [$\text{cm}^5 \text{V}^{-1} \text{s}^{-1}$]
dmdt	Derivative of μ_{dop} with respect to t [$\text{cm}^2 \text{V}^{-1} \text{s}^{-1} \text{K}^{-1}$]

In most cases, it is not necessary to compute the derivatives with respect to the dopant concentrations.

However, to model random dopant fluctuations (see [Random Dopant Fluctuations on page 503](#)), the PMI model must override the functions that compute the following values:

dmdNa	Derivative of μ_{dop} with respect to the acceptor concentration [$\text{cm}^5 \text{V}^{-1} \text{s}^{-1}$]
dmdNd	Derivative of μ_{dop} with respect to the donor concentration [$\text{cm}^5 \text{V}^{-1} \text{s}^{-1}$]

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_DopingDepMobility : public PMI_Vertex_Interface {

private:
    const PMI_AnisotropyType anisoType;

public:
    PMI_DopingDepMobility (const PMI_Environment& env,
                           const PMI_AnisotropyType anisotype);
    virtual ~PMI_DopingDepMobility () ;

    PMI_AnisotropyType AnisotropyType () const { return anisoType; }

    virtual void Compute_m
        (const double n, const double p,
         const double t, double& m) = 0;

    virtual void Compute_dmdn
        (const double n, const double p,
         const double t, double& dmdn) = 0;

    virtual void Compute_dmdp
        (const double n, const double p,
```

34: Physical Model Interface

Doping-dependent Mobility

```
    const double t, double& dmdp) = 0;

    virtual void Compute_dmdt
        (const double n, const double p,
         const double t, double& dmdt) = 0;

    virtual void Compute_dmdNa
        (const double n, const double p,
         const double t, double& dmdNa);

    virtual void Compute_dmdNd
        (const double n, const double p,
         const double t, double& dmdNd);

};

};

Two virtual constructors are required for electron and hole mobilities:
```

```
typedef PMI_DopingDepMobility* new_PMI_DopingDepMobility_func
    (const PMI_Environment& env, const PMI_AnisotropyType anisotype);
extern "C" new_PMI_DopingDepMobility_func new_PMI_DopingDep_e_Mobility;
extern "C" new_PMI_DopingDepMobility_func new_PMI_DopingDep_h_Mobility;
```

Example: Masetti Model

The built-in Masetti model (see [Masetti Model on page 269](#)) can also be implemented as a PMI model:

```
#include "PMIModels.h"

class Masetti_DopingDepMobility : public PMI_DopingDepMobility {
protected:
    const double T0;
    double mumax, Exponent, mumini1, mumini2, mul, Pc, Cr, Cs, alpha, beta;

public:
    Masetti_DopingDepMobility (const PMI_Environment& env,
                               const PMI_AnisotropyType anisotype);
    ~Masetti_DopingDepMobility () {}

    void Compute_m
        (const double n, const double p,
         const double t, double& m);

    void Compute_dmdn
        (const double n, const double p,
         const double t, double& dmdn);
```

```

void Compute_dmdp
  (const double n, const double p,
   const double t, double& dmdp);

void Compute_dmdt
  (const double n, const double p,
   const double t, double& dmdt);
};

Masetti_DopingDepMobility::
Masetti_DopingDepMobility (const PMI_Environment& env,
                           const PMI_AnisotropyType anisotype) :
  PMI_DopingDepMobility (env, anisotype),
  T0 (300.0)
{
}

void Masetti_DopingDepMobility:::
Compute_m (const double n, const double p,
           const double t, double& m)
{ const double mu_const = mumax * pow (t/T0, -Exponent);
  const double Ni = Max (ReadDoping (PMI_Donor) +
                         ReadDoping (PMI_Acceptor), 1.0);
  m = mumin1 * exp (-Pc / Ni) +
       (mu_const - mumin2) / (1.0 + pow (Ni / Cr, alpha)) -
       mul / (1.0 + pow (Cs / Ni, beta));
}

void Masetti_DopingDepMobility:::
Compute_dmdn (const double n, const double p,
               const double t, double& dmdn)
{ dmdn = 0.0;
}

void Masetti_DopingDepMobility:::
Compute_dmdp (const double n, const double p,
               const double t, double& dmdp)
{ dmdp = 0.0;
}

void Masetti_DopingDepMobility:::
Compute_dmdt (const double n, const double p,
               const double t, double& dmdt)
{ const double Ni = Max (ReadDoping (PMI_Donor) +
                         ReadDoping (PMI_Acceptor), 1.0);
  dmdt = mumax * (-Exponent/T0) * pow (t/T0, -Exponent - 1.0) /

```

34: Physical Model Interface

Doping-dependent Mobility

```
        (1.0 + pow (Ni / Cr, alpha));
    }

class Masetti_e_DopingDepMobility : public Masetti_DopingDepMobility {
public:
    Masetti_e_DopingDepMobility (const PMI_Environment& env,
                                const PMI_AnisotropyType anisotype);
    ~Masetti_e_DopingDepMobility () {}
};

Masetti_e_DopingDepMobility:::
Masetti_e_DopingDepMobility (const PMI_Environment& env,
                            const PMI_AnisotropyType anisotype) :
    Masetti_DopingDepMobility (env, anisotype)

{ // default values
    mumax = InitParameter ("mumax_e", 1417.0);
    Exponent = InitParameter ("Exponent_e", 2.5);
    mumin1 = InitParameter ("mumin1_e", 52.2);
    mumin2 = InitParameter ("mumin2_e", 52.2);
    mul = InitParameter ("mul_e", 43.4);
    Pc = InitParameter ("Pc_e", 0.0);
    Cr = InitParameter ("Cr_e", 9.68e16);
    Cs = InitParameter ("Cs_e", 3.43e20);
    alpha = InitParameter ("alpha_e", 0.680);
    beta = InitParameter ("beta_e", 2.0);
}

class Masetti_h_DopingDepMobility : public Masetti_DopingDepMobility {
public:
    Masetti_h_DopingDepMobility (const PMI_Environment& env,
                                const PMI_AnisotropyType anisotype);
    ~Masetti_h_DopingDepMobility () {}
};

Masetti_h_DopingDepMobility:::
Masetti_h_DopingDepMobility (const PMI_Environment& env,
                            const PMI_AnisotropyType anisotype) :
    Masetti_DopingDepMobility (env, anisotype)

{ // default values
    mumax = InitParameter ("mumax_h", 470.5);
    Exponent = InitParameter ("Exponent_h", 2.2);
    mumin1 = InitParameter ("mumin1_h", 44.9);
    mumin2 = InitParameter ("mumin2_h", 0.0);
    mul = InitParameter ("mul_h", 29.0);
    Pc = InitParameter ("Pc_h", 9.23e16);
    Cr = InitParameter ("Cr_h", 2.23e17);
```

```
Cs = InitParameter ("Cs_h", 6.10e20);
alpha = InitParameter ("alpha_h", 0.719);
beta = InitParameter ("beta_h", 2.0);
}

extern "C"
PMI_DopingDepMobility* new_PMI_DopingDep_e_Mobility
  (const PMI_Environment& env, const PMI_AnisotropyType anisotype)
{ return new Masetti_e_DopingDepMobility (env, anisotype);
}

extern "C"
PMI_DopingDepMobility* new_PMI_DopingDep_h_Mobility
  (const PMI_Environment& env, const PMI_AnisotropyType anisotype)
{ return new Masetti_h_DopingDepMobility (env, anisotype);
}
```

Multistate Configuration-dependent Bulk Mobility

This PMI allows you to implement multistate configuration (MSC)-dependent bulk mobility models.

Command File

To activate a PMI of this type, as an option to `eMobility`, `hMobility`, or `Mobility` in the `Physics` section, specify:

```
DopingDependence(
  PMIModel (
    Name = <string>
    MSConfig = <string>
    Index = <int>
    String = <string>
  )
)
```

The options of `PMIModel` are described in [Command File on page 991](#).

34: Physical Model Interface

Multistate Configuration–dependent Bulk Mobility

Dependencies

The mobility may depend on the variables:

n	Electron density [cm ⁻³]
p	Hole density [cm ⁻³]
T	Lattice temperature [K]
eT	Electron temperature [K]
hT	Hole temperature [K]
s	Multistate configuration occupation probabilities [1]

The model must compute the following quantities:

val	Mobility μ_{dop} [cm ² V ⁻¹ s ⁻¹]
dval_dn	Derivative with respect to electron density [cm ⁵ V ⁻¹ s ⁻¹]
dval_dp	Derivative with respect to hole density [cm ⁵ V ⁻¹ s ⁻¹]
dval_dT	Derivative with respect to lattice temperature [cm ² V ⁻¹ s ⁻¹ K ⁻¹]
dval_deT	Derivative with respect to electron temperature [cm ² V ⁻¹ s ⁻¹ K ⁻¹]
dval_dhT	Derivative with respect to hole temperature [cm ² V ⁻¹ s ⁻¹ K ⁻¹]
dval_ds	Derivative with respect to multistate configuration occupation probabilities [cm ² V ⁻¹ s ⁻¹]

C++ Interface

The PMI offers a base class that presents the following interface:

```
class PMI_MSC_Mobility : public PMI_MSC_Vertex_Interface
{
public:
    PMI_MSC_Mobility (const PMI_Environment& env,
                      const std::string& msconfig_name,
                      const int model_index,
                      const std::string& model_string,
```

```
    const PMI_AnisotropyType aniso);
    // otherwise, see C++ Interface on page 993
};
```

Apart from the name of the base class and the constructor, the explanations in [C++ Interface on page 993](#) apply here as well.

The following virtual constructor must be implemented:

```
typedef PMI_MSC_Mobility* new_PMI_MSC_Mobility_func
(const PMI_Environment& env, const std::string& msconfig_name,
 int model_index, const std::string& model_string,
 const PMI_AnisotropyType anisotype);
extern "C" new_PMI_MSC_Mobility_func new_PMI_MSC_Mobility;
```

Mobility Degradation at Interfaces

Sentaurus Device uses Mathiessen's rule:

$$\frac{1}{\mu} = \frac{1}{\mu_{\text{dop}}} + \frac{1}{\mu_{\text{enormal}}} \quad (970)$$

to combine the constant and doping-dependent mobility μ_{dop} , and the surface contribution μ_{enormal} (see [Mobility Degradation at Interfaces on page 273](#)). To express no mobility degradation, for example, in the bulk of a device, it is necessary to set $\mu_{\text{enormal}} = \infty$. To avoid numeric difficulties, the PMI requires the calculation of the inverse mobility $1/\mu_{\text{enormal}}$ instead of μ_{enormal} .

As an additional precaution, Sentaurus Device does not evaluate the PMI model if the normal electric field F_{\perp} is less than `EnormMinimum`, where `EnormMinimum` is a parameter that can be specified in the `PMI_model_name` section of the parameter file. By default, `EnormMinimum = 1 V/cm` is used.

Dependencies

The mobility degradation at interfaces may depend on the following variables:

<code>dist</code>	Distance to nearest interface [cm]
<code>pot</code>	Electrostatic potential [V]
<code>enorm</code>	Normal electric field [Vcm^{-1}]

34: Physical Model Interface

Mobility Degradation at Interfaces

t	Lattice temperature [K]
n	Electron density [cm^{-3}]
p	Hole density [cm^{-3}]
ct	Carrier temperature [K]

NOTE If Sentaurus Device cannot determine the distance to the nearest interface, the value of $\text{dist} = 10^{10}$ is used.

NOTE The carrier temperature ct represents the electron temperature during the evaluation of the model for electrons, and the hole temperature during the evaluation of the model for holes. The parameter ct is only defined for hydrodynamic simulations. Otherwise, the value of $\text{ct} = 0$ is used.

The PMI model must compute the following results:

muinv	Inverse of mobility $1/\mu_{\text{enormal}}$ [cm^{-2}Vs]
dmuinvdpot	Derivative of $1/\mu_{\text{enormal}}$ with respect to pot [cm^{-2}s]
dmuinvdenorm	Derivative of $1/\mu_{\text{enormal}}$ with respect to enorm [cm^{-1}s]
dmuinvdn	Derivative of $1/\mu_{\text{enormal}}$ with respect to n [cmVs]
dmuinvdp	Derivative of $1/\mu_{\text{enormal}}$ with respect to p [cmVs]
dmuinvdt	Derivative of $1/\mu_{\text{enormal}}$ with respect to t [$\text{cm}^{-2}\text{VsK}^{-1}$]
dmuinvdct	Derivative of $1/\mu_{\text{enormal}}$ with respect to ct [$\text{cm}^{-2}\text{VsK}^{-1}$]

In most cases, it is not necessary to compute the derivatives with respect to the dopant concentrations. However, to model random dopant fluctuations (see [Random Dopant Fluctuations on page 503](#)), the PMI model must override the functions that compute the following values:

dmuinvdNa	Derivative of $1/\mu_{\text{enormal}}$ with respect to the acceptor concentration [cmVs]
dmuinvdNd	Derivative of $1/\mu_{\text{enormal}}$ with respect to the donor concentration [cmVs]

C++ Interface

The enumeration type `PMI_EnormalType` describes the type of the normal electric field F_{\perp} :

```
enum PMI_EnormalType {
    PMI_EnormalToCurrent,
    PMI_EnormalToInterface
};
```

The following base class is declared in the file `PMIModels.h`:

```
class PMI_EnormalMobility : public PMI_Vertex_Interface {

private:
    const PMI_EnormalType enormalType;
    const PMI_AnisotropyType anisoType;

public:
    PMI_EnormalMobility (const PMI_Environment& env,
                         const PMI_EnormalType type,
                         const PMI_AnisotropyType anisotype);

    virtual ~PMI_EnormalMobility () ;

    PMI_EnormalType EnormalType () const { return enormalType; }
    PMI_AnisotropyType AnisotropyType () const { return anisoType; }

    virtual void Compute_muinv
        (const double dist, const double pot,
         const double enorm, const double n, const double p,
         const double t, const double ct, double& muinv) = 0;

    virtual void Compute_dmuinvdpot
        (const double dist, const double pot,
         const double enorm, const double n, const double p,
         const double t, const double ct, double& dmuinvdpot) = 0;

    virtual void Compute_dmuinvdenorm
        (const double dist, const double pot,
         const double enorm, const double n, const double p,
         const double t, const double ct, double& dmuinvdenorm) = 0;

    virtual void Compute_dmuinvdn
        (const double dist, const double pot,
         const double enorm, const double n, const double p,
         const double t, const double ct, double& dmuinvdn) = 0;

    virtual void Compute_dmuinvdp
```

34: Physical Model Interface

Mobility Degradation at Interfaces

```
(const double dist, const double pot,
  const double enorm, const double n, const double p,
  const double t, const double ct, double& dmuinvdp) = 0;

virtual void Compute_dmuinvdt
  (const double dist, const double pot,
  const double enorm, const double n, const double p,
  const double t, const double ct, double& dmuinvdt) = 0;

virtual void Compute_dmuinvdct
  (const double dist, const double pot,
  const double enorm, const double n, const double p,
  const double t, const double ct, double& dmuinvdct) = 0;

virtual void Compute_dmuinvdNa
  (const double dist, const double pot,
  const double enorm, const double n, const double p,
  const double t, const double ct, double& dmuinvdNa);

virtual void Compute_dmuinvdNd
  (const double dist, const double pot,
  const double enorm, const double n, const double p,
  const double t, const double ct, double& dmuinvdNd);

};

};

Two virtual constructors are required for electron and hole mobilities:
```

```
typedef PMI_EnormalMobility* new_PMI_EnormalMobility_func
  (const PMI_Environment& env, const PMI_EnormalType type,
   const PMI_AnisotropyType anisotype);
extern "C" new_PMI_EnormalMobility_func new_PMI_Enormal_e_Mobility;
extern "C" new_PMI_EnormalMobility_func new_PMI_Enormal_h_Mobility;
```

Example: Lombardi Model

This example illustrates the implementation of a slightly simplified Lombardi model (see [Mobility Degradation at Interfaces on page 273](#)) using the PMI. The contribution due to acoustic phonon-scattering has the form:

$$\mu_{ac} = \frac{B}{F_\perp} + \frac{CN_i^\lambda}{F_\perp^{1/3}(T/T_0)} \quad (971)$$

where $T_0 = 300$ K.

The contribution due to surface roughness scattering is given by:

$$\mu_{sr} = \left(\frac{F_\perp^2}{\delta} + \frac{F_\perp^3}{\eta} \right)^{-1} \quad (972)$$

The mobilities μ_{ac} and μ_{sr} are combined according to Mathiessen's rule with an additional damping factor:

$$\frac{1}{\mu_{enormal}} = e^{-\frac{l}{l_{crit}}} \cdot \left(\frac{1}{\mu_{ac}} + \frac{1}{\mu_{sr}} \right) \quad (973)$$

where l is the distance to the nearest semiconductor-insulator interface point:

```
#include "PMIModels.h"

class Lombardi_EnormalMobility : public PMI_EnormalMobility {

protected:
    const double T0;
    double B, C, lambda, delta, eta, l_crit;

public:
    Lombardi_EnormalMobility (const PMI_Environment& env,
                             const PMI_EnormalType type,
                             const PMI_AnisotropyType anisotype);

~Lombardi_EnormalMobility () ;

    void Compute_muinv
        (const double dist, const double pot,
         const double enorm, const double n, const double p,
         const double t, const double ct, double& muinv);

    void Compute_dmuinvdpot
        (const double dist, const double pot,
         const double enorm, const double n, const double p,
         const double t, const double ct, double& dmuinvdpot);

    void Compute_dmuinvdenorm
        (const double dist, const double pot,
         const double enorm, const double n, const double p,
         const double t, const double ct, double& dmuinvdenorm);

    void Compute_dmuinvdn
        (const double dist, const double pot,
         const double enorm, const double n, const double p,
         const double t, const double ct, double& dmuinvdn);
}
```

34: Physical Model Interface

Mobility Degradation at Interfaces

```
void Compute_dmuinvdp
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdp);

void Compute_dmuinvdt
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdt);

void Compute_dmuinvdct
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdct);
};

Lombardi_EnormalMobility::
Lombardi_EnormalMobility (const PMI_Environment& env,
                           const PMI_EnormalType type,
                           const PMI_AnisotropyType anisotype) :
    PMI_EnormalMobility (env, type, anisotype),
    T0 (300.0)
{
}

Lombardi_EnormalMobility::
~Lombardi_EnormalMobility ()
{
}

void Lombardi_EnormalMobility::
Compute_muinv (const double dist, const double pot,
               const double enorm, const double n, const double p,
               const double t, const double ct, double& muinv)
{ const double Ni = ReadDoping (PMI_Donor) + ReadDoping (PMI_Acceptor);
  const double denom_ac_inv =
      B + pow (enorm, 2.0/3.0) * C * pow (Ni, lambda) * T0 / t;
  const double mu_ac_inv = enorm / denom_ac_inv;
  const double mu_sr_inv = enorm * enorm / delta + pow (enorm, 3.0) / eta;
  const double damping = exp (-dist/l_crit);
  muinv = damping * (mu_ac_inv + mu_sr_inv);
}

void Lombardi_EnormalMobility::
Compute_dmuinvdpot (const double dist, const double pot,
                     const double enorm, const double n, const double p,
```

```

        const double t, const double ct, double& dmuinvdpot)
{ dmuinvdpot = 0.0;
}

void Lombardi_EnormalMobility::
Compute_dmuinvdenorm (const double dist, const double pot,
                      const double enorm, const double n, const double p,
                      const double t, const double ct, double& dmuinvdenorm)
{ const double Ni = ReadDoping (PMI_Donor) + ReadDoping (PMI_Acceptor);
  const double denom_ac_inv =
    B + pow (enorm, 2.0/3.0) * C * pow (Ni, lambda) * T0 / t;
  const double dmu_ac_inv_denorm =
    (2.0 * B + denom_ac_inv) / (3.0 * denom_ac_inv * denom_ac_inv);
  const double mu_sr_inv_denorm =
    2.0 * enorm / delta + 3.0 * enorm * enorm / eta;
  const double damping = exp (-dist/l_crit);
  dmuinvdenorm = damping * (dmu_ac_inv_denorm + mu_sr_inv_denorm);
}

void Lombardi_EnormalMobility::
Compute_dmuinvdn (const double dist, const double pot,
                  const double enorm, const double n, const double p,
                  const double t, const double ct, double& dmuinvdn)
{ dmuinvdn = 0.0;
}

void Lombardi_EnormalMobility::
Compute_dmuinvdp (const double dist, const double pot,
                  const double enorm, const double n, const double p,
                  const double t, const double ct, double& dmuinvdp)
{ dmuinvdp = 0.0;
}

void Lombardi_EnormalMobility::
Compute_dmuinvdt (const double dist, const double pot,
                  const double enorm, const double n, const double p,
                  const double t, const double ct, double& dmuinvdt)
{ const double Ni = ReadDoping (PMI_Donor) + ReadDoping (PMI_Acceptor);
  const double factor = pow (enorm, 2.0/3.0) * C * pow (Ni, lambda) * T0;
  const double denom_ac_inv = B + factor / t;
  const double dmu_ac_inv_dt =
    enorm * factor / (denom_ac_inv * denom_ac_inv * t * t);
  const double damping = exp (-dist/l_crit);
  dmuinvdt = damping * dmu_ac_inv_dt;
}

void Lombardi_EnormalMobility::
Compute_dmuinvdct (const double dist, const double pot,

```

34: Physical Model Interface

Mobility Degradation at Interfaces

```
        const double enorm, const double n, const double p,
        const double t, const double ct, double& dmuinvdct)
{ dmuinvdct = 0.0;
}

class Lombardi_e_EnormalMobility : public Lombardi_EnormalMobility {
public:
    Lombardi_e_EnormalMobility (const PMI_Environment& env,
                                const PMI_EnormalType type,
                                const PMI_AnisotropyType anisotype);

    ~Lombardi_e_EnormalMobility () {}
};

Lombardi_e_EnormalMobility:::
Lombardi_e_EnormalMobility (const PMI_Environment& env,
                            const PMI_EnormalType type,
                            const PMI_AnisotropyType anisotype) :
    Lombardi_EnormalMobility (env, type, anisotype)
{ // default values
    B = InitParameter ("B_e", 4.750e7);
    C = InitParameter ("C_e", 580.0);
    lambda = InitParameter ("lambda_e", 0.125);
    delta = InitParameter ("delta_e", 5.82e14);
    eta = InitParameter ("eta_e", 5.82e30);
    l_crit = InitParameter ("l_crit_e", 1.0e-6);
}

class Lombardi_h_EnormalMobility : public Lombardi_EnormalMobility {
public:
    Lombardi_h_EnormalMobility (const PMI_Environment& env,
                                const PMI_EnormalType type,
                                const PMI_AnisotropyType anisotype);

    ~Lombardi_h_EnormalMobility () {}
};

Lombardi_h_EnormalMobility:::
Lombardi_h_EnormalMobility (const PMI_Environment& env,
                            const PMI_EnormalType type,
                            const PMI_AnisotropyType anisotype) :
    Lombardi_EnormalMobility (env, type, anisotype)
{ // default values
    B = InitParameter ("B_h", 9.925e6);
    C = InitParameter ("C_h", 2947.0);
    lambda = InitParameter ("lambda_h", 0.0317);
    delta = InitParameter ("delta_h", 2.0546e14);
    eta = InitParameter ("eta_h", 2.0546e30);
```

```

    l_crit = InitParameter ("l_crit_h", 1.0e-6);
}

extern "C"
PMI_EnormalMobility* new_PMI_Enormal_e_Mobility
    (const PMI_Environment& env, const PMI_EnormalType type,
     const PMI_AnisotropyType anisotype)
{ return new Lombardi_e_EnormalMobility (env, type, anisotype);
}

extern "C"
PMI_EnormalMobility* new_PMI_Enormal_h_Mobility
    (const PMI_Environment& env, const PMI_EnormalType type,
     const PMI_AnisotropyType anisotype)
{ return new Lombardi_h_EnormalMobility (env, type, anisotype);
}

```

High-Field Saturation Model

The high-field saturation model computes the final mobility μ as a function of the low-field mobility μ_{low} and the driving force F_{hfs} (see [High-Field Saturation on page 294](#)).

Dependencies

The mobility μ computed by a high-field mobility model may depend on the following variables:

pot	Electrostatic potential [V]
t	Lattice temperature [K]
n	Electron density [cm^{-3}]
p	Hole density [cm^{-3}]
ct	Carrier temperature [K]
μ_{low}	Low-field mobility μ_{low} [$\text{cm}^2 \text{V}^{-1} \text{s}^{-1}$]
F	Driving force [Vcm^{-1}]

34: Physical Model Interface

High-Field Saturation Model

NOTE The carrier temperature ct represents the electron temperature during the evaluation of the model for electrons, and the hole temperature during the evaluation of the model for holes. The parameter ct is only defined for hydrodynamic simulations. Otherwise, the value of $ct = 0$ is used.

The PMI model must compute the following results:

μ	Mobility μ [$\text{cm}^2\text{V}^{-1}\text{s}^{-1}$]
$d\mu/dpot$	Derivative of μ with respect to pot [$\text{cm}^2\text{V}^{-2}\text{s}^{-1}$]
$d\mu/dn$	Derivative of μ with respect to n [$\text{cm}^5\text{V}^{-1}\text{s}^{-1}$]
$d\mu/dp$	Derivative of μ with respect to p [$\text{cm}^5\text{V}^{-1}\text{s}^{-1}$]
$d\mu/dt$	Derivative of μ with respect to t [$\text{cm}^2\text{V}^{-1}\text{s}^{-1}\text{K}^{-1}$]
$d\mu/dct$	Derivative of μ with respect to ct [$\text{cm}^2\text{V}^{-1}\text{s}^{-1}\text{K}^{-1}$]
$d\mu/dmulow$	Derivative of μ with respect to $mulow$ (1)
$d\mu/dF$	Derivative of μ with respect to F [$\text{cm}^3\text{V}^{-2}\text{s}^{-1}$]

In most cases, it is not necessary to compute the derivatives with respect to the dopant concentrations. However, to model random dopant fluctuations (see [Random Dopant Fluctuations on page 503](#)), the PMI model must override the functions that compute the following values:

$d\mu/dNa$	Derivative of μ with respect to the acceptor concentration [$\text{cm}^5\text{V}^{-1}\text{s}^{-1}$]
$d\mu/dNd$	Derivative of μ with respect to the donor concentration [$\text{cm}^5\text{V}^{-1}\text{s}^{-1}$]

C++ Interface

The enumeration type `PMI_HighFieldDrivingForce` describes the driving force as specified in the command file:

```
enum PMI_HighFieldDrivingForce {
    PMI_HighFieldParallelElectricField,
    PMI_HighFieldParallelToInterfaceElectricField,
    PMI_HighFieldGradQuasiFermi
};
```

The following base class is declared in the file PMIModels.h:

```
class PMI_HighFieldMobility : public PMI_Vertex_Interface {

private:
    const PMI_HighFieldDrivingForce drivingForce;
    const PMI_AnisotropyType anisoType;

public:
    PMI_HighFieldMobility (const PMI_Environment& env,
                           const PMI_HighFieldDrivingForce force,
                           const PMI_AnisotropyType anisotype);

    virtual ~PMI_HighFieldMobility () ;

    PMI_HighFieldDrivingForce HighFieldDrivingForce () const { return
drivingForce; }
    PMI_AnisotropyType AnisotropyType () const { return anisoType; }

    virtual void Compute_mu
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& mu) = 0;

    virtual void Compute_dmudpot
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudpot) = 0;

    virtual void Compute_dmudn
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudn) = 0;

    virtual void Compute_dmudp
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudp) = 0;

    virtual void Compute_dmudt
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudt) = 0;

    virtual void Compute_dmudct
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudct) = 0;
```

34: Physical Model Interface

High-Field Saturation Model

```
virtual void Compute_dmudmulow
  (const double pot, const double n,
   const double p, const double t, const double ct,
   const double mulow, const double F, double& dmudmulow) = 0;

virtual void Compute_dmudF
  (const double pot, const double n,
   const double p, const double t, const double ct,
   const double mulow, const double F, double& dmudF) = 0;

virtual void Compute_dmudNa
  (const double pot, const double n,
   const double p, const double t, const double ct,
   const double mulow, const double F, double& dmudNa);

virtual void Compute_dmudNd
  (const double pot, const double n,
   const double p, const double t, const double ct,
   const double mulow, const double F, double& dmudNd);
};


```

Two virtual constructors are required for electron and hole mobilities:

```
typedef PMI_HighFieldMobility* new_PMI_HighFieldMobility_func
  (const PMI_Environment& env, const PMI_HighFieldDrivingForce force,
   const PMI_AnisotropyType anisotype);
extern "C" new_PMI_HighFieldMobility_func new_PMI_HighField_e_Mobility;
extern "C" new_PMI_HighFieldMobility_func new_PMI_HighField_h_Mobility;
```

Example: Canali Model

This example presents the PMI implementation of the Canali model:

$$\mu = \frac{\mu_{\text{low}}}{\left[1 + \left(\frac{\mu_{\text{low}} F_{\text{hfs}}}{v_{\text{sat}}} \right)^{\beta} \right]^{1/\beta}} \quad (974)$$

where:

$$\beta = \beta_0 \left(\frac{T}{T_0} \right)^{\beta_{\text{exp}}} \quad (975)$$

and:

$$v_{\text{sat}} = v_{\text{sat0}} \left(\frac{T}{T_0} \right)^{v_{\text{sat,exp}}} \quad (976)$$

The built-in Canali model is discussed in [Extended Canali Model on page 295](#).

```
#include "PMIModels.h"

class Canali_HighFieldMobility : public PMI_HighFieldMobility {

private:
    double beta, vsat, Fabs, val, valb, valb1, valb11b;
    void Compute_internal (const double t, const double mulow,
                           const double F);

protected:
    const double T0;
    double beta0, betaexp, vsat0, vsatexp;

public:
    Canali_HighFieldMobility (const PMI_Environment& env,
                             const PMI_HighFieldDrivingForce force,
                             const PMI_AnisotropyType anisotype);

~Canali_HighFieldMobility ();

void Compute_mu
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F,
     double& mu);

void Compute_dmudpot
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudpot);

void Compute_dmudn
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudn);

void Compute_dmudp
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudp);

void Compute_dmudt
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudt);

void Compute_dmudct
```

34: Physical Model Interface

High-Field Saturation Model

```
(const double pot, const double n,
 const double p, const double t, const double ct,
 const double mulow, const double F, double& dmudct);

void Compute_dmudmulow
(const double pot, const double n,
 const double p, const double t, const double ct,
 const double mulow, const double F, double& dmudmulow);

void Compute_dmudF
(const double pot, const double n,
 const double p, const double t, const double ct,
 const double mulow, const double F, double& dmudF);
};

void Canali_HighFieldMobility::
Compute_internal (const double t, const double mulow, const double F)
{ beta = beta0 * pow (t/T0, betaexp);
  vsat = vsat0 * pow (t/T0, -vsatexp);
  Fabs = fabs (F);
  val = mulow * Fabs / vsat;
  valb = pow (val, beta);
  valb1 = 1.0 + valb;
  valb11b = pow (valb1, 1.0/beta);
}

Canali_HighFieldMobility::
Canali_HighFieldMobility (const PMI_Environment& env,
                         const PMI_HighFieldDrivingForce force,
                         const PMI_AnisotropyType anisotype) :
  PMI_HighFieldMobility (env, force, anisotype),
  T0 (300.0)
{
}

Canali_HighFieldMobility::
~Canali_HighFieldMobility ()
{
}

void Canali_HighFieldMobility::
Compute_mu (const double pot, const double n,
            const double p, const double t, const double ct,
            const double mulow, const double F, double& mu)
{ Compute_internal (t, mulow, F);
  mu = mulow / valb11b;
}
```

```

void Canali_HighFieldMobility::
Compute_dmudpot (const double pot, const double n,
                  const double p, const double t, const double ct,
                  const double mulow, const double F, double& dmudpot)
{ dmudpot = 0.0;
}

void Canali_HighFieldMobility::
Compute_dmudn (const double pot, const double n,
                const double p, const double t, const double ct,
                const double mulow, const double F, double& dmudn)
{ dmudn = 0.0;
}

void Canali_HighFieldMobility::
Compute_dmudp (const double pot, const double n,
                const double p, const double t, const double ct,
                const double mulow, const double F, double& dmudp)
{ dmudp = 0.0;
}

void Canali_HighFieldMobility::
Compute_dmudt (const double pot, const double n,
                const double p, const double t, const double ct,
                const double mulow, const double F, double& dmudt)
{ Compute_internal (t, mulow, F);
  const double mu = mulow / valb1b;
  const double dmudbeta = mu * (log (valb1) / (beta*beta) -
                                 valb * log (val) / (beta * valb1));
  const double dmudvsat = (mu * valb) / (valb1 * vsat);
  const double dbetadt = beta * betaxp / t;
  const double dvsatdt = -vsat * vsatexp / t;
  dmudt = dmudbeta * dbetadt + dmudvsat * dvsatdt;
}

void Canali_HighFieldMobility::
Compute_dmudct (const double pot, const double n,
                 const double p, const double t, const double ct,
                 const double mulow, const double F, double& dmudct)
{ dmudct = 0.0;
}

void Canali_HighFieldMobility::
Compute_dmudmulow (const double pot, const double n,
                   const double p, const double t, const double ct,
                   const double mulow, const double F, double& dmudmulow)
{ Compute_internal (t, mulow, F);
}

```

34: Physical Model Interface

High-Field Saturation Model

```
dmudmulow = 1.0 / (valb1 * valb1lb);
}

void Canali_HighFieldMobility::  
Compute_dmudF (const double pot, const double n,  
                const double p, const double t, const double ct,  
                const double mulow, const double F, double& dmudF)  
{ Compute_internal (t, mulow, F);  
    const double mu = mulow / valb1lb;  
    const double signF = (F >= 0.0) ? 1.0 : -1.0;  
    dmudF = -mu * pow (mulow/vsat, beta) * pow (fabs, beta-1.0) *  
            signF / valb1;  
}  
  
class Canali_e_HighFieldMobility : public Canali_HighFieldMobility {  
public:  
    Canali_e_HighFieldMobility (const PMI_Environment& env,  
                               const PMI_HighFieldDrivingForce force,  
                               const PMI_AnisotropyType anisotype);  
  
    ~Canali_e_HighFieldMobility () {}  
};  
  
Canali_e_HighFieldMobility::  
Canali_e_HighFieldMobility (const PMI_Environment& env,  
                           const PMI_HighFieldDrivingForce force,  
                           const PMI_AnisotropyType anisotype) :  
    Canali_HighFieldMobility (env, force, anisotype)  
{ // default values  
    beta0 = InitParameter ("beta0_e", 1.109);  
    betaexp = InitParameter ("betaexp_e", 0.66);  
    vsat0 = InitParameter ("vsat0_e", 1.07e7);  
    vsatexp = InitParameter ("vsatexp_e", 0.87);  
}  
  
class Canali_h_HighFieldMobility : public Canali_HighFieldMobility {  
public:  
    Canali_h_HighFieldMobility (const PMI_Environment& env,  
                               const PMI_HighFieldDrivingForce force,  
                               const PMI_AnisotropyType anisotype);  
  
    ~Canali_h_HighFieldMobility () {}  
};  
  
Canali_h_HighFieldMobility::  
Canali_h_HighFieldMobility (const PMI_Environment& env,  
                           const PMI_HighFieldDrivingForce force,  
                           const PMI_AnisotropyType anisotype) :
```

```

Canali_HighFieldMobility (env, force, anisotype)
{ // default values
    beta0 = InitParameter ("beta0_h", 1.213);
    betaexp = InitParameter ("betaexp_h", 0.17);
    vsat0 = InitParameter ("vsat0_h", 8.37e6);
    vsatexp = InitParameter ("vsatexp_h", 0.52);
}

extern "C"
PMI_HighFieldMobility* new_PMI_HighField_e_Mobility
    (const PMI_Environment& env, const PMI_HighFieldDrivingForce force,
     const PMI_AnisotropyType anisotype)
{ return new Canali_e_HighFieldMobility (env, force, anisotype);
}

extern "C"
PMI_HighFieldMobility* new_PMI_HighField_h_Mobility
    (const PMI_Environment& env, const PMI_HighFieldDrivingForce force,
     const PMI_AnisotropyType anisotype)
{ return new Canali_h_HighFieldMobility (env, force, anisotype);
}

```

Band Gap

Sentaurus Device provides a PMI to compute the energy band gap E_g in a semiconductor. It can be specified in the `Physics` section of the command file, for example:

```

Physics {
    EffectiveIntrinsicDensity (
        BandGap (pmi_model_name)
    )
}

```

The default bandgap model in Sentaurus Device is selected explicitly by the keyword `Default`:

```

Physics {
    EffectiveIntrinsicDensity (
        BandGap (Default)
    )
}

```

34: Physical Model Interface

Band Gap

Dependencies

The band gap E_g may depend on:

t Lattice temperature [K]

The PMI model must compute the following results:

bg	Band gap E_g [eV]
dbgdt	Derivative of bg with respect to t [eVK ⁻¹]

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_BandGap : public PMI_Vertex_Interface {  
  
public:  
    PMI_BandGap (const PMI_Environment& env);  
    virtual ~PMI_BandGap ();  
  
    virtual void Compute_bg  
        (const double t, double& bg) = 0;  
  
    virtual void Compute_dbgdt  
        (const double t, double& dbgdt) = 0;  
};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_BandGap* new_PMI_BandGap_func  
    (const PMI_Environment& env);  
extern "C" new_PMI_BandGap_func new_PMI_BandGap;
```

Example: Default Bandgap Model

Sentaurus Device uses the following default bandgap model:

$$E_g(t) = E_g(0) - \frac{\alpha t^2}{t + \beta} \quad (977)$$

$E_g(0)$ denotes the band gap at 0 K.

```
#include "PMIModels.h"

class Default_BandGap : public PMI_BandGap {

private:
    double Eg0, alpha, beta;

public:
    Default_BandGap (const PMI_Environment& env);

    ~Default_BandGap () ;

    void Compute_bg (const double t, double& bg);

    void Compute_dbgdt (const double t, double& dbgdt);
};

Default_BandGap::
Default_BandGap (const PMI_Environment& env) :
    PMI_BandGap (env)
{ Eg0 = InitParameter ("Eg0", 1.16964);
  alpha = InitParameter ("alpha", 4.73e-4);
  beta = InitParameter ("beta", 636);
}

Default_BandGap::
~Default_BandGap ()
{
}

void Default_BandGap::
Compute_bg (const double t, double& bg)
{ bg = Eg0 - alpha * t * t / (t + beta);
}

void Default_BandGap::
Compute_dbgdt (const double t, double& dbgdt)
{ dbgdt = - alpha * t * (t + 2.0 * beta) / ((t + beta) * (t + beta));
}
```

34: Physical Model Interface

Bandgap Narrowing

```
}
```

```
extern "C"
```

```
PMI_BandGap* new_PMI_BandGap
```

```
(const PMI_Environment& env)
```

```
{ return new Default_BandGap (env);
```

```
}
```

Bandgap Narrowing

Sentaurus Device provides a PMI to compute bandgap narrowing (see [Band Gap and Electron Affinity on page 225](#)). A user model is activated with the keyword `EffectiveIntrinsicDensity` in the `Physics` section of the command file:

```
Physics {
```

```
    EffectiveIntrinsicDensity (pmi_model_name)
```

```
}
```

Dependencies

A PMI bandgap narrowing model has no explicit dependencies. However, it can depend on doping concentrations through the run-time support.

The PMI model must compute:

`bgn` Bandgap narrowing ΔE_g^0 [eV]

In most cases, it is not necessary to compute the derivatives with respect to the dopant concentrations. However, to model random dopant fluctuations (see [Random Dopant Fluctuations on page 503](#)), the PMI model must override the functions that compute the following values:

`dbgndNa` Derivative of ΔE_g^0 with respect to the acceptor concentration [cm³eV]

`dbgndNd` Derivative of ΔE_g^0 with respect to the donor concentration [cm³eV]

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_BandGapNarrowing : public PMI_Vertex_Interface {
public:
    PMI_BandGapNarrowing (const PMI_Environment& env);
    virtual ~PMI_BandGapNarrowing ();
    virtual void Compute_bgn (double& bgn) = 0;
    virtual void Compute_dbgndNa (double& dbgndNa);
    virtual void Compute_dbgndNd (double& dbgndNd);
};
```

The following virtual constructor must be implemented:

```
typedef PMI_BandGapNarrowing* new_PMI_BandGapNarrowing_func
(const PMI_Environment& env);
extern "C" new_PMI_BandGapNarrowing_func new_PMI_BandGapNarrowing;
```

Example: Default Model

The default bandgap narrowing model in Sentaurus Device (Bennett–Wilson) is given by:

$$\Delta E_g^0 = \begin{cases} E_{\text{ref}} \left[\ln \frac{N_{\text{tot}}}{N_{\text{ref}}} \right]^2, & N_{\text{tot}} > N_{\text{ref}}, \\ 0, & N_{\text{tot}} \leq N_{\text{ref}}. \end{cases} \quad (978)$$

See [Band Gap and Electron Affinity on page 225](#).

This model can be implemented as a PMI model as follows:

```
#include "PMIModels.h"

class Bennett_BandGapNarrowing : public PMI_BandGapNarrowing {

private:
    double Ebgn, Nref;

public:
    Bennett_BandGapNarrowing (const PMI_Environment& env);

    ~Bennett_BandGapNarrowing ();

    void Compute_bgn (double& bgn);
```

34: Physical Model Interface

Apparent Band-Edge Shift

```
};

Bennett_BandGapNarrowing::  
Bennett_BandGapNarrowing (const PMI_Environment& env) :  
    PMI_BandGapNarrowing (env)  
{ Ebgn = InitParameter ("Ebgn", 6.84e-3);  
    Nref = InitParameter ("Nref", 3.162e18);  
}  
  
Bennett_BandGapNarrowing::  
~Bennett_BandGapNarrowing ()  
{  
}  
  
void Bennett_BandGapNarrowing::  
Compute_bgn (double& bgn)  
{ const double Na = ReadDoping (PMI_Acceptor);  
    const double Nd = ReadDoping (PMI_Donor);  
    const double Ni = Na + Nd;  
    if (Ni > Nref) {  
        const double tmp = log (Ni / Nref);  
        bgn = Ebgn * tmp * tmp;  
    } else {  
        bgn = 0.0;  
    }  
}  
  
extern "C"  
PMI_BandGapNarrowing* new_PMI_BandGapNarrowing  
    (const PMI_Environment& env)  
{ return new Bennett_BandGapNarrowing (env);  
}
```

Apparent Band-Edge Shift

The apparent band-edge shift Λ_{PMI} is a quantity similar to bandgap narrowing. In contrast to bandgap narrowing, the apparent band-edge shift can depend on the solution variables (electron and hole densities, lattice temperature, and electric field). Conversely, the apparent band-edge shift does not take effect in all situations where a real band-edge shift takes effect (this is why the band-edge shift is called ‘apparent’).

Implementationwise, the apparent band-edge shift is an extension of the density gradient model (see [Density Gradient Quantization Model on page 260](#)). For the PMI model, this implies:

- Sentaurus Device applies the apparent band-edge shift Λ_{PMI} everywhere where it applies quantization corrections.
- By default, the apparent band-edge shift that Sentaurus Device computes is not equal to Λ_{PMI} , but contains contributions from quantization. To remove them, set $\gamma = 0$ (see [Density Gradient Quantization Model on page 260](#)).
- Apart from a specification in the `Physics` section, it is necessary to specify additional equations in the `Solve` section (see [Using the Density Gradient Model on page 261](#)).

To select a model to compute Λ_{PMI} , specify its name using the `LocalModel` keyword (see [Table 208 on page 1166](#)). The same models for Λ_{PMI} can be used for the shift of the conduction and valence bands. A positive value of Λ_{PMI} means that the band shifts outwards, away from midgap (therefore, the band gap widens).

Dependencies

The apparent band-edge shift Λ_{PMI} may depend on:

n	Electron density [cm ⁻³]
p	Hole density [cm ⁻³]
t	Lattice temperature [K]
F	Absolute value of the electric field [Vcm ⁻¹]

The PMI model must compute the following values:

shift	Apparent band-edge shift Λ_{PMI} [eV]
dshiftdn	Derivative of Λ_{PMI} with respect to n [eVcm ³]
dshiftdp	Derivative of Λ_{PMI} with respect to p [eVcm ³]
dshiftdt	Derivative of Λ_{PMI} with respect to t [eVK ⁻¹]
dshiftdf	Derivative of Λ_{PMI} with respect to f [eVcmV ⁻¹]

34: Physical Model Interface

Apparent Band-Edge Shift

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_ApparentBandEdgeShift : public PMI_Vertex_Interface {  
  
public:  
    PMI_ApparentBandEdgeShift (const PMI_Environment& env);  
    virtual ~PMI_ApparentBandEdgeShift () ;  
  
    virtual void Compute_shift  
        (const double n, const double p,  
         const double t, const double f,  
         double& shift) = 0;  
  
    virtual void Compute_dshiftdn  
        (const double n, const double p,  
         const double t, const double f,  
         double& dshiftdn) = 0;  
  
    virtual void Compute_dshiftdp  
        (const double n, const double p,  
         const double t, const double f,  
         double& dshiftdp) = 0;  
  
    virtual void Compute_dshiftdt  
        (const double n, const double p,  
         const double t, const double f,  
         double& dshiftdt) = 0;  
  
    virtual void Compute_dshiftdf  
        (const double n, const double p,  
         const double t, const double f,  
         double& dshiftdf) = 0;  
};
```

The following virtual constructor must be implemented:

```
typedef PMI_ApparentBandEdgeShift* new_PMI_ApparentBandEdgeShift_func  
    (const PMI_Environment& env);  
extern "C" new_PMI_ApparentBandEdgeShift_func new_PMI_ApparentBandEdgeShift;
```

Multistate Configuration-dependent Apparent Band-Edge Shift

The multistate configuration (MSC)-dependent, apparent band-edge shift model is a variant of the apparent band-edge shift model (see [Apparent Band-Edge Shift on page 970](#)). Besides dependencies on the solution variables, electron and hole densities, lattice temperature, and carrier temperatures, it allows dependencies on all state occupation rates of an arbitrary reference MSC defined by `MSConfig`. The remarks made for the apparent band-edge shift in connection with the density gradient model are valid here as well.

The model can be selected as arguments of the keywords `eBandEdgeShift`, `hBandEdgeShift`, and `BandEdgeShift` (see [Apparent Band-Edge Shift on page 368](#)) in the `MSConfig` specification. The optional model index parameter allows you to implement, in the same model, several variants that can be accessed from the command file.

Dependencies

The MSC apparent band-edge shift Λ_{PMI} may depend on:

n	Electron density [cm ⁻³]
p	Hole density [cm ⁻³]
t	Lattice temperature [K]
ct	Carrier temperature [K]
s	Vector of state occupation rates of the reference MSC [1]

The PMI model must compute the following values if the dependency is used:

shift	Apparent band-edge shift Λ_{PMI} [eV]
dshiftdn	Derivative of Λ_{PMI} with respect to n [eVcm ³]
dshiftdp	Derivative of Λ_{PMI} with respect to p [eVcm ³]
dshiftdt	Derivative of Λ_{PMI} with respect to t [eVK ⁻¹]
dshiftdf	Derivative of Λ_{PMI} with respect to ct [eVK ⁻¹]
dshiftds	Derivative of Λ_{PMI} with respect to s [eV]

Additional Functionality

The PMI model provides additional functionality.

Using Dependencies

You can use the function `set_dependency_used` to switch on or off the dependencies of this model explicitly (the default is on). For used dependencies, the function computing the corresponding derivative must be provided; for unused dependencies, the functions are not called.

Updating Actual Status

Before calling the computation functions (`compute_val` and `compute_dval_dx`), the simulator passes the actual values of the dependencies to the model using `set_actual_status`. The model parameters are updated by `init_parameter` before the actual status is updated.

C++ Interface

The following base class (here, only an extract) is declared in the file `PMIModels.h`:

```
class PMI_MSC_ApparentBandEdgeShift : public PMI_MSC_Vertex_Interface {  
  
public:  
    enum e_var { var_n, var_p, var_T, var_eT, var_hT, var_s, var_undefined };  
  
    class input_data {  
    public:  
        input_data ();  
        ~input_data ();  
        double& val ( e_var var, size_t ind );  
        double val ( e_var var, size_t ind ) const;  
    };  
  
    public:  
        PMI_MSC_ApparentBandEdgeShift (const PMI_Environment& env,  
            const std::string& msconfig_name, int model_index = 0);  
        virtual ~PMI_MSC_ApparentBandEdgeShift ();  
  
        // get names of MSConfig and its states  
        const std::string& msconfig_name () const;  
        size_t nb_states () const;
```

```

const std::string& state ( size_t index ) const;

virtual void set_actual_status (
    const PMI_MSC_ApparentBandEdgeShift::input_data& id );

// compute value and derivatives
virtual void compute_val ( double& val );
virtual void compute_dval_dn ( double& val );
virtual void compute_dval_dp ( double& val );
virtual void compute_dval_dT ( double& val );
virtual void compute_dval_deT ( double& val );
virtual void compute_dval_dhT ( double& val );
virtual void compute_dval_ds ( std::vector<double>& val );

// support ramping of parameters
virtual void init_parameter ();

// handle dependencies
void set_dependency_used (
    PMI_MSC_ApparentBandEdgeShift::e_var var, bool flag );
bool dependency_used ( PMI_MSC_ApparentBandEdgeShift::e_var var ) const;
};


```

The following virtual constructor must be implemented:

```

typedef PMI_MSC_ApparentBandEdgeShift* new_PMI_MSC_ApparentBandEdgeShift_func
    (const PMI_Environment& env,
     const std::string& msconfig_name,
     int model_index);
extern "C" new_PMI_ApparentBandEdgeShift_func
new_PMI_MSC_ApparentBandEdgeShift;

```

Electron Affinity

The electron affinity χ , that is, the energy separation between the conduction band and vacuum level, can be specified by using a PMI. The syntax in the command file is:

```

Physics {
    Affinity (pmi_model_name)
}

```

The default affinity model in Sentaurus Device can be selected explicitly by the keyword Default:

```

Physics {
    Affinity (Default)
}

```

34: Physical Model Interface

Electron Affinity

Dependencies

The electron affinity χ may depend on:

t Lattice temperature [K]

The PMI model must compute:

`affinity` Electron affinity χ [eV]

`affinitydt` Derivative of `affinity` with respect to t [eVK^{-1}]

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Affinity : public PMI_Vertex_Interface {  
  
public:  
    PMI_Affinity (const PMI_Environment& env);  
    virtual ~PMI_Affinity ();  
  
    virtual void Compute_affinity  
        (const double t, double& affinity) = 0;  
  
    virtual void Compute_daffinitydt  
        (const double t, double& daffinitydt) = 0;  
};
```

The prototype for the virtual constructor is:

```
typedef PMI_Affinity* new_PMI_Affinity_func  
    (const PMI_Environment& env);  
extern "C" new_PMI_Affinity_func new_PMI_Affinity;
```

Example: Default Affinity Model

By default, Sentaurus Device uses this formula to compute χ :

$$\chi(t) = \chi(0) + 0.5 \frac{\alpha t^2}{t + \beta} \quad (979)$$

$\chi(0)$ denotes the affinity at 0 K.

```
#include "PMIModels.h"

class Default_Affinity : public PMI_Affinity {

private:
    double Affinity0, alpha, beta;

public:
    Default_Affinity (const PMI_Environment& env);

    ~Default_Affinity ();

    void Compute_affinity (const double t, double& affinity);

    void Compute_daffinitydt (const double t, double& daffinitydt);

};

Default_Affinity::
Default_Affinity (const PMI_Environment& env) :
    PMI_Affinity (env)
{ Affinity0 = InitParameter ("Affinity0", 4.05);
  alpha = InitParameter ("alpha", 4.73e-4);
  beta = InitParameter ("beta", 636);
}

Default_Affinity::
~Default_Affinity ()
{
}

void Default_Affinity::
Compute_affinity (const double t, double& affinity)
{ affinity = Affinity0 + 0.5 * alpha * t * t / (t + beta);
}

void Default_Affinity::
Compute_daffinitydt (const double t, double& daffinitydt)
```

34: Physical Model Interface

Effective Mass

```
{ daffinitydt = 0.5 * alpha * t * (t + 2.0 * beta) /  
    ((t + beta) * (t + beta));  
}  
extern "C"  
PMI_Affinity* new_PMI_Affinity  
    (const PMI_Environment& env)  
{ return new Default_Affinity (env);  
}
```

Effective Mass

Sentaurus Device provides a PMI to compute the effective mass of electrons and holes. The effective mass is always expressed as a multiple of the electron mass in vacuum. The name of the PMI model must appear in the Physics section of the command file:

```
Physics {  
    EffectiveMass (pmi_model_name)  
}
```

Dependencies

The relative effective mass may depend on the following variables:

t	Lattice temperature [K]
bg	Band gap [eV]

The PMI model must compute the following results:

m	Relative effective mass (1)
dm/dt	Derivative of m with respect to t [K^{-1}]
dm/dbg	Derivative of m with respect to bg [eV^{-1}]

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_EffectiveMass : public PMI_Vertex_Interface {

public:
    PMI_EffectiveMass (const PMI_Environment& env);
    virtual ~PMI_EffectiveMass () ;

    virtual void Compute_m
        (const double t, const double bg, double& m) = 0;

    virtual void Compute_dmdt
        (const double t, const double bg, double& dmdt) = 0;

    virtual void Compute_dmdbg
        (const double t, const double bg, double& dmdbg) = 0;
};
```

Two virtual constructors are necessary to compute the effective mass of electrons and holes:

```
typedef PMI_EffectiveMass* new_PMI_EffectiveMass_func
    (const PMI_Environment& env);
extern "C" new_PMI_EffectiveMass_func new_PMI_e_EffectiveMass;
extern "C" new_PMI_EffectiveMass_func new_PMI_h_EffectiveMass;
```

Example: Linear Effective Mass Model

A simple, linear effective mass model is given by:

$$m = m_{300} + \frac{dm}{dt}(t - 300) \quad (980)$$

m_{300} denotes the mass at 300 K. It can be implemented as follows:

```
#include "PMIModels.h"

class Linear_EffectiveMass : public PMI_EffectiveMass {

protected:
    double mass_300, dmass_dt;

public:
    Linear_EffectiveMass (const PMI_Environment& env);
```

34: Physical Model Interface

Effective Mass

```
~Linear_EffectiveMass () ;

void Compute_m (const double t, const double bg, double& m) ;

void Compute_dmdt (const double t, const double bg, double& dmdt) ;

void Compute_dmdbg (const double t, const double bg, double& dmdbg) ;
};

Linear_EffectiveMass::
Linear_EffectiveMass (const PMI_Environment& env) :
    PMI_EffectiveMass (env)
{
}

Linear_EffectiveMass::
~Linear_EffectiveMass ()
{
}

void Linear_EffectiveMass::
Compute_m (const double t, const double bg, double& m)
{ m = mass_300 + dmass_dt * (t - 300.0);
}

void Linear_EffectiveMass::
Compute_dmdt (const double t, const double bg, double& dmdt)
{ dmdt = dmass_dt;
}

void Linear_EffectiveMass::
Compute_dmdbg (const double t, const double bg, double& dmdbg)
{ dmdbg = 0.0;
}

class Linear_e_EffectiveMass : public Linear_EffectiveMass {

public:
    Linear_e_EffectiveMass (const PMI_Environment& env);

    ~Linear_e_EffectiveMass () {}

};

Linear_e_EffectiveMass::
Linear_e_EffectiveMass (const PMI_Environment& env) :
    Linear_EffectiveMass (env)
{ mass_300 = InitParameter ("mass_e_300", 1.09);
```

```

    dmass_dt = InitParameter ("dmass_e_dt", 1.6e-4);
}

class Linear_h_EffectiveMass : public Linear_EffectiveMass {

public:
    Linear_h_EffectiveMass (const PMI_Environment& env);

    ~Linear_h_EffectiveMass () {}

};

Linear_h_EffectiveMass::Linear_h_EffectiveMass (const PMI_Environment& env) :
    Linear_EffectiveMass (env)
{ mass_300 = InitParameter ("mass_h_300", 1.15);
    dmass_dt = InitParameter ("dmass_h_dt", 9.2e-4);
}

extern "C"
PMI_EffectiveMass* new_PMI_e_EffectiveMass
    (const PMI_Environment& env)
{ return new Linear_e_EffectiveMass (env);
}

extern "C"
PMI_EffectiveMass* new_PMI_h_EffectiveMass
    (const PMI_Environment& env)
{ return new Linear_h_EffectiveMass (env);
}

```

Energy Relaxation Times

The model for the energy relaxation times τ in [Eq. 51, p. 187](#) and [Eq. 52, p. 187](#) can be specified in the Physics section of the command file. The four available possibilities are:

```

Physics {
    EnergyRelaxationTimes (
        formula
        constant
        irrational
        pmi_model_name
    )
}

```

34: Physical Model Interface

Energy Relaxation Times

These entries have the following meaning:

formula	Use the value of formula in the parameter file (default)
constant	Use constant energy relaxation times (formula = 1)
irrational	Use the ratio of two irrational polynomials (formula = 2)
pmi_model_name	Call a PMI model to compute the energy relaxation times

Dependencies

The energy relaxation time τ may depend on the variable:

ct Carrier temperature [K]

NOTE The parameter ct represents the electron temperature during the calculation of τ_n and the hole temperature during the calculation of τ_p .

The PMI model must compute the following results:

tau Energy relaxation time τ [s]

dtaudct Derivative of τ with respect to ct [sK^{-1}]

C++ Interface

The following base class is declared in the file PMIModels.h:

```
class PMI_EnergyRelaxationTime : public PMI_Vertex_Interface {  
  
public:  
    PMI_EnergyRelaxationTime (const PMI_Environment& env);  
  
    virtual ~PMI_EnergyRelaxationTime ();  
  
    virtual void Compute_tau  
        (const double ct, double& tau) = 0;  
  
    virtual void Compute_dtaudct  
        (const double ct, double& dtaudct) = 0;  
  
};
```

The following two virtual constructors must be implemented for electron and hole energy relaxation times:

```
typedef PMI_EnergyRelaxationTime* new_PMI_EnergyRelaxationTime_func  
    (const PMI_Environment& env);  
extern "C" new_PMI_EnergyRelaxationTime_func new_PMI_e_EnergyRelaxationTime;  
extern "C" new_PMI_EnergyRelaxationTime_func new_PMI_h_EnergyRelaxationTime;
```

Example: Constant Energy Relaxation Times

The following C++ code implements constant energy relaxation times:

```
#include "PMIModels.h"  
  
class Const_EnergyRelaxationTime : public PMI_EnergyRelaxationTime {  
  
protected:  
    double tau_const;  
  
public:  
    Const_EnergyRelaxationTime (const PMI_Environment& env);  
  
    ~Const_EnergyRelaxationTime ();  
  
    void Compute_tau  
        (const double ct, double& tau);  
  
    void Compute_dtaudct  
        (const double ct, double& dtaudct);  
  
};  
  
Const_EnergyRelaxationTime::  
Const_EnergyRelaxationTime (const PMI_Environment& env) :  
    PMI_EnergyRelaxationTime (env)  
{  
}  
  
Const_EnergyRelaxationTime::  
~Const_EnergyRelaxationTime ()  
{  
}  
  
void Const_EnergyRelaxationTime::  
Compute_tau (const double ct, double& tau)  
{ tau = tau_const;  
}
```

34: Physical Model Interface

Energy Relaxation Times

```
void Const_EnergyRelaxationTime::  
Compute_dtaudct (const double ct, double& dtaudct)  
{ dtaudct = 0.0;  
}  
  
class Const_e_EnergyRelaxationTime : public Const_EnergyRelaxationTime {  
  
public:  
    Const_e_EnergyRelaxationTime (const PMI_Environment& env);  
  
    ~Const_e_EnergyRelaxationTime () {}  
  
};  
  
Const_e_EnergyRelaxationTime::  
Const_e_EnergyRelaxationTime (const PMI_Environment& env) :  
    Const_EnergyRelaxationTime (env)  
{ tau_const = InitParameter ("tau_const_e", 0.3e-12);  
}  
  
class Const_h_EnergyRelaxationTime : public Const_EnergyRelaxationTime {  
  
public:  
    Const_h_EnergyRelaxationTime (const PMI_Environment& env);  
  
    ~Const_h_EnergyRelaxationTime () {}  
  
};  
  
Const_h_EnergyRelaxationTime::  
Const_h_EnergyRelaxationTime (const PMI_Environment& env) :  
    Const_EnergyRelaxationTime (env)  
{ tau_const = InitParameter ("tau_const_h", 0.25e-12);  
}  
  
extern "C"  
PMI_EnergyRelaxationTime* new_PMI_e_EnergyRelaxationTime  
    (const PMI_Environment& env)  
{ return new Const_e_EnergyRelaxationTime (env);  
}  
  
extern "C"  
PMI_EnergyRelaxationTime* new_PMI_h_EnergyRelaxationTime  
    (const PMI_Environment& env)  
{ return new Const_h_EnergyRelaxationTime (env);  
}
```

Lifetimes

This PMI provides access to the electron and hole lifetimes, τ_n and τ_p , in the SRH recombination (see [Eq. 260, p. 292](#)) and the coupled defect level (CDL) recombination (see [Eq. 289, p. 315](#)). In the command file, the names of the lifetime models are given as arguments to the SRH or CDL keywords:

```
Physics {
    Recombination (SRH (pmi_model_name))
}
```

or:

```
Physics {
    Recombination (CDL (pmi_model_name))
}
```

NOTE A PMI model overrides all other keywords in an SRH or a CDL statement.

Dependencies

A PMI lifetime model may depend on the variable:

t Lattice temperature [K]

It must compute the following results:

τ Lifetime τ [s]

$d\tau/dt$ Derivative of τ with respect to lattice temperature [sK^{-1}]

C++ Interface

The enumeration type `PMI_LifetimeModel` describes where the PMI lifetime is used:

```
enum PMI_LifetimeModel {
    PMI_SRH,
    PMI_CDL1,
    PMI_CDL2
};
```

34: Physical Model Interface

Lifetimes

The following base class is declared in the file PMIModels.h:

```
class PMI_Lifetime : public PMI_Vertex_Interface {  
  
private:  
    const PMI_LifetimeModel lifetimeModel;  
  
public:  
    PMI_Lifetime (const PMI_Environment& env,  
                  const PMI_LifetimeModel model);  
  
    virtual ~PMI_Lifetime ();  
  
    PMI_LifetimeModel LifetimeModel () const { return lifetimeModel; }  
  
    virtual void Compute_tau  
        (const double t, double& tau) = 0;  
  
    virtual void Compute_dtaudt  
        (const double t, double& dtaudt) = 0;  
};
```

Two virtual constructors must be implemented for electron and hole lifetimes:

```
typedef PMI_Lifetime* new_PMI_Lifetime_func  
    (const PMI_Environment& env, const PMI_LifetimeModel model);  
extern "C" new_PMI_Lifetime_func new_PMI_e_Lifetime;  
extern "C" new_PMI_Lifetime_func new_PMI_h_Lifetime;
```

Example: Doping- and Temperature-dependent Lifetimes

The following example combines doping-dependent lifetimes (Scharfetter) and temperature dependence (power law):

$$\tau = \left(\tau_{\min} + \frac{\tau_{\max} - \tau_{\min}}{1 + \left(\frac{N_{A,0} + N_{D,0}}{N_{\text{ref}}} \right)^{\gamma}} \right) \left(\frac{T}{300\text{K}} \right)^{\alpha} \quad (981)$$

```
#include "PMIModels.h"  
  
class Scharfetter_Lifetime : public PMI_Lifetime {  
  
protected:  
    const double T0;  
    double taumin, taumax, Nref, gamma, Talpha;
```

```

public:
    Scharfetter_Lifetime (const PMI_Environment& env,
                          const PMI_LifetimeModel model);

    ~Scharfetter_Lifetime ();
    void Compute_tau
        (const double t, double& tau);

    void Compute_dtaudt
        (const double t, double& dtaudt);

};

Scharfetter_Lifetime::
Scharfetter_Lifetime (const PMI_Environment& env,
                      const PMI_LifetimeModel model) :
    PMI_Lifetime (env, model),
    T0 (300.0)
{
}

Scharfetter_Lifetime::
~Scharfetter_Lifetime ()
{
}

void Scharfetter_Lifetime::
Compute_tau (const double t, double& tau)
{ const double Ni = ReadDoping (PMI_Acceptor) + ReadDoping (PMI_Donor);
  tau = taumin + (taumax - taumin) / (1.0 + pow (Ni/Nref, gamma));
  tau *= pow (t/T0, Talpha);
}

void Scharfetter_Lifetime::
Compute_dtaudt (const double t, double& dtaudt)
{ const double Ni = ReadDoping (PMI_Acceptor) + ReadDoping (PMI_Donor);
  dtaudt = taumin + (taumax - taumin) / (1.0 + pow (Ni/Nref, gamma));
  dtaudt *= (Talpha/T0) * pow (t/T0, Talpha-1.0);
}

class Scharfetter_e_Lifetime : public Scharfetter_Lifetime {

public:
    Scharfetter_e_Lifetime (const PMI_Environment& env,
                           const PMI_LifetimeModel model);

    ~Scharfetter_e_Lifetime () {}
}

```

34: Physical Model Interface

Lifetimes

```
};

Scharfetter_e_Lifetime::
Scharfetter_e_Lifetime (const PMI_Environment& env,
                       const PMI_LifetimeModel model) :
    Scharfetter_Lifetime (env, model)
{ taumin = InitParameter ("taumin_e", 0.0);
  taumax = InitParameter ("taumax_e", 1.0e-5);
  Nref = InitParameter ("Nref_e", 1.0e16);
  gamma = InitParameter ("gamma_e", 1.0);
  Talpha = InitParameter ("Talpha_e", -1.5);
}

class Scharfetter_h_Lifetime : public Scharfetter_Lifetime {

public:
    Scharfetter_h_Lifetime (const PMI_Environment& env,
                           const PMI_LifetimeModel model);

    ~Scharfetter_h_Lifetime () {}

};

Scharfetter_h_Lifetime::
Scharfetter_h_Lifetime (const PMI_Environment& env,
                       const PMI_LifetimeModel model) :
    Scharfetter_Lifetime (env, model)

{ taumin = InitParameter ("taumin_h", 0.0);
  taumax = InitParameter ("taumax_h", 3.0e-6);
  Nref = InitParameter ("Nref_h", 1.0e16);
  gamma = InitParameter ("gamma_h", 1.0);
  Talpha = InitParameter ("Talpha_h", -1.5);
}

extern "C"
PMI_Lifetime* new_PMI_e_Lifetime
  (const PMI_Environment& env, const PMI_LifetimeModel model)
{ return new Scharfetter_e_Lifetime (env, model);
}

extern "C"
PMI_Lifetime* new_PMI_h_Lifetime
  (const PMI_Environment& env, const PMI_LifetimeModel model)
{ return new Scharfetter_h_Lifetime (env, model);
}
```

Thermal Conductivity

The PMI provides access to the lattice thermal conductivity κ in [Eq. 32, p. 180](#). To activate it, in the Physics section of the command file, specify:

```
Physics {
    ThermalConductivity (
        <string>
    )
}
```

where the string is the name of the PMI model.

The PMI supports anisotropic thermal conductivity, and the model can be evaluated along different crystallographic axes. The enumeration type `PMI_AnisotropyType` as defined in [Mobility Models on page 942](#) determines the axis. The default is isotropic thermal conductivity. If anisotropic thermal conductivity is activated in the command file, the PMI class `PMI_ThermalConductivity` is also instantiated in the anisotropic direction.

Dependencies

The thermal conductivity κ may depend on the variable:

t Lattice temperature [K]

The PMI model must compute the following results:

κ Thermal conductivity κ [$\text{Wcm}^{-1}\text{K}^{-1}$]
 $d\kappa/dt$ derivative of κ with respect to t [$\text{Wcm}^{-1}\text{K}^{-2}$]

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_ThermalConductivity : public PMI_Vertex_Interface {

public:
    PMI_ThermalConductivity (const PMI_Environment& env, const
    PMI_AnisotropyType anisotype);
```

34: Physical Model Interface

Thermal Conductivity

```
virtual ~PMI_ThermalConductivity () ;

PMI_AnisotropyType AnisotropyType () const { return anisoType; }

virtual void Compute_kappa
    (const double t, double& kappa) = 0;

virtual void Compute_dkappadt
    (const double t, double& dkappadt) = 0;

};
```

The following virtual constructor must be implemented:

```
typedef PMI_ThermalConductivity* new_PMI_ThermalConductivity_func
    (const PMI_Environment& env, const PMI_AnisotropyType anisotype);
extern "C" new_PMI_ThermalConductivity_func
new_PMI_ThermalConductivity;
```

Example: Temperature-dependent Thermal Conductivity

The following C++ code implements the temperature-dependent thermal conductivity:

$$\kappa(T) = \frac{1}{a + bT + cT^2} \quad (982)$$

as given in [Eq. 808, p. 788](#).

```
#include "PMIModels.h"

class TempDep_ThermalConductivity : public PMI_ThermalConductivity {
private:
    double a, b, c;

public:
    TempDep_ThermalConductivity (const PMI_Environment& env, const
        PMI_AnisotropyType anisotype);
    ~TempDep_ThermalConductivity () ;

    void Compute_kappa
        (const double t, double& kappa);

    void Compute_dkappadt
        (const double t, double& dkappadt);
};
```

```

TempDep_ThermalConductivity::  

TempDep_ThermalConductivity (const PMI_Environment& env, const  

PMI_AnisotropyType anisotype) :  

    PMI_ThermalConductivity (env, anisotype)  

{ // default values  

    a = InitParameter ("a", 0.03);  

    b = InitParameter ("b", 1.56e-03);  

    c = InitParameter ("c", 1.65e-06);  

}  
  

TempDep_ThermalConductivity::  

~TempDep_ThermalConductivity ()  

{  

}  
  

void TempDep_ThermalConductivity::  

Compute_kappa (const double t, double& kappa)  

{ kappa = 1.0 / (a + b*t + c*t*t);  

}  
  

void TempDep_ThermalConductivity::  

Compute_dkappadt (const double t, double& dkappadt)  

{ const double kappa = 1.0 / (a + b*t + c*t*t);  

    dkappadt = -kappa * kappa * (b + 2.0*c*t);  

}  
  

extern "C"  

PMI_ThermalConductivity* new_PMI_ThermalConductivity  

    (const PMI_Environment& env, const PMI_AnisotropyType anisotype)  

{ return new TempDep_ThermalConductivity (env, anisotype);  

}

```

Multistate Configuration–dependent Thermal Conductivity

This PMI provides access to the lattice thermal conductivity κ in [Eq. 32, p. 180](#) and allows it to depend on a multistate configuration (see [Chapter 11 on page 365](#)).

Command File

To activate a PMI of this type, in the Physics section, specify:

```

ThermalConductivity(
    PMIModel (
        Name = <string>

```

34: Physical Model Interface

Multistate Configuration–dependent Thermal Conductivity

```
MSConfig = <string>
Index = <int>
String = <string>
)
)
```

Here, `Name` is the name of the PMI model; its specification is mandatory. `MSConfig` selects the name of the multistate configuration. `MSConfig` defaults to an empty string, which means the model does not depend on any multistate configuration. `Index` and `String` are optional; they determine the arguments `model_index` and `model_string` that are passed to the virtual constructor (see below); the interpretation of those arguments is up to the PMI model. `Index` defaults to zero, and `String` defaults to the empty string.

Dependencies

The thermal conductivity may depend on the variables:

n	Electron density [cm ⁻³]
p	Hole density [cm ⁻³]
T	Lattice temperature [K]
eT	Electron temperature [K]
hT	Hole temperature [K]
s	Multistate configuration occupation probabilities [1]

The model must compute the following quantities:

val	Thermal conductivity [Wcm ⁻¹ K ⁻¹]
dval_dn	Derivative with respect to electron density [Wcm ² K ⁻¹]
dval_dp	Derivative with respect to hole density [Wcm ² K ⁻¹]
dval_dT	Derivative with respect to lattice temperature [Wcm ⁻¹ K ⁻²]
dval_deT	Derivative with respect to electron temperature [Wcm ⁻¹ K ⁻²]
dval_dhT	Derivative with respect to hole temperature [Wcm ⁻¹ K ⁻²]
dval_ds	Derivative with respect to multistate configuration occupation probabilities [Wcm ⁻¹ K ⁻¹]

C++ Interface

The PMI offers a base class that presents the following interface:

```

class PMI_MSC_ThermalConductivity : public PMI_MSC_Vertex_Interface
{
public:
    class idata {
    public:
        double n () const;
        double p () const;
        double T () const;
        double eT () const;
        double hT () const;
        double s (size_t ind) const;
    };

    class odata {
    public:
        double& val ();
        double& dval_dn ();
        double& dval_dp ();
        double& dval_dT ();
        double& dval_det ();
        double& dval_dht ();
        double& dval_ds ( size_t ind );
    };

    PMI_MSC_ThermalConductivity(const PMI_Environment& env,
        const std::string& msconfig_name,
        const int model_index,
        const std::string& model_string,
        const PMI_AnisotropyType anisotype);
    virtual ~PMI_MSC_ThermalConductivity () ;

    PMI_AnisotropyType AnisotropyType () const;

    virtual void compute
        (const idata* id,
         odata* od ) = 0;
    };
}

```

The `compute` function receives its input from `id`. It returns the results using `od` by assignment using the member functions of `od`. The PMI framework initializes the values of the derivatives to zero, so you do not have to do anything for derivatives with respect to the variables on which your model does not depend.

34: Physical Model Interface

Heat Capacity

The following virtual constructor must be implemented:

```
typedef PMI_MSC_ThermalConductivity* new_PMI_MSC_ThermalConductivity_func
(const PMI_Environment& env, const std::string& msconfig_name,
 int model_index, const std::string& model_string,
 const PMI_AnisotropyType anisotype);
extern "C" new_PMI_MSC_ThermalConductivity_func
new_PMI_MSC_ThermalConductivity;
```

Heat Capacity

The model for lattice heat capacity in [Eq. 32, p. 180](#) can be specified in the Physics section of the command file. The following two possibilities are available:

```
Physics {
    HeatCapacity (
        constant
        pmi_model_name
    )
}
```

These entries have the following meaning:

- | | |
|----------------|---|
| constant | Use constant heat capacity (default) |
| pmi_model_name | Call a PMI model to compute the heat capacity |

Dependencies

The heat capacity c_L may depend on the variable:

- | | |
|---|-------------------------|
| t | Lattice temperature [K] |
|---|-------------------------|

The PMI model must compute the following results:

- | | |
|------|--|
| c | Heat capacity c_L [$\text{JK}^{-1}\text{cm}^{-3}$] |
| dcdt | Derivative of c_L with respect to t [$\text{JK}^{-2}\text{cm}^{-3}$] |

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_HeatCapacity : public PMI_Vertex_Interface {  
  
public:  
    PMI_HeatCapacity (const PMI_Environment& env);  
  
    virtual ~PMI_HeatCapacity ();  
  
    virtual void Compute_c  
        (const double t, double& c) = 0;  
  
    virtual void Compute_dcdt  
        (const double t, double& dcdt) = 0  
  
};
```

The following virtual constructor must be implemented:

```
typedef PMI_HeatCapacity* new_PMI_HeatCapacity_func  
    (const PMI_Environment& env);  
extern "C" new_PMI_HeatCapacity_func new_PMI_HeatCapacity;
```

Example: Constant Heat Capacity

The following C++ code implements constant heat capacity:

```
#include "PMIModels.h"  
  
class Constant_HeatCapacity : public PMI_HeatCapacity {  
private:  
    double cv;  
  
public:  
    Constant_HeatCapacity (const PMI_Environment& env);  
    ~Constant_HeatCapacity ();  
  
    void Compute_c  
        (const double t, double& c);  
  
    void Compute_dcdt  
        (const double t, double& dcdt);  
};
```

34: Physical Model Interface

Multistate Configuration–dependent Heat Capacity

```
Constant_HeatCapacity::  
Constant_HeatCapacity (const PMI_Environment& env) :  
    PMI_HeatCapacity (env)  
{ // default values  
    cv = InitParameter ("cv", 1.63);  
}  
  
Constant_HeatCapacity::  
~Constant_HeatCapacity ()  
{  
}  
  
void Constant_HeatCapacity::  
Compute_c (const double t, double& c)  
{ c = cv;  
}  
  
void Constant_HeatCapacity::  
Compute_dcdt (const double t, double& dcdt)  
{ dcdt = 0.0;  
}  
  
extern "C"  
PMI_HeatCapacity* new_PMI_HeatCapacity  
    (const PMI_Environment& env)  
{ return new Constant_HeatCapacity (env);  
}
```

Multistate Configuration–dependent Heat Capacity

This PMI computes the lattice heat capacity and allows it to depend on a multistate configuration (see [Chapter 11 on page 365](#)).

Command File

To activate a PMI of this type, in the Physics section, specify:

```
HeatCapacity(  
    PMIModel (  
        Name = <string>  
        MSConfig = <string>
```

```

    Index = <int>
    String = <string>
)
)

```

The options of **PMIModel** are described in [Command File on page 991](#).

Dependencies

The heat capacity may depend on the variables:

n	Electron density [cm ⁻³]
p	Hole density [cm ⁻³]
T	Lattice temperature [K]
eT	Electron temperature [K]
hT	Hole temperature [K]
s	Multistate configuration occupation probabilities [1]

The model must compute the following quantities:

val	Heat capacity [Jcm ⁻³ K ⁻¹]
dval_dn	Derivative with respect to electron density [JK ⁻¹]
dval_dp	Derivative with respect to hole density [JK ⁻¹]
dval_dT	Derivative with respect to lattice temperature [Jcm ⁻³ K ⁻²]
dval_deT	Derivative with respect to electron temperature [Jcm ⁻³ K ⁻²]
dval_dhT	Derivative with respect to hole temperature [Jcm ⁻³ K ⁻²]
dval_ds	Derivative with respect to multistate configuration occupation probabilities [Jcm ⁻³ K ⁻¹]

C++ Interface

The PMI offers a base class that presents the following interface:

```
class PMI_MSC_HeatCapacity : public PMI_MSC_Vertex_Interface
{
public:
    PMI_MSC_HeatCapacity (const PMI_Environment& env,
                          const std::string& msconfig_name,
                          const int model_index,
                          const std::string& model_string);
    // otherwise, see C++ Interface on page 993
};
```

Apart from the base class name and the constructor, the explanations in [C++ Interface on page 993](#) apply here as well.

The following virtual constructor must be implemented:

```
typedef PMI_MSC_HeatCapacity* new_PMI_MSC_HeatCapacity_func
(const PMI_Environment& env, const std::string& msconfig_name,
 int model_index, const std::string& model_string);
extern "C" new_PMI_MSC_HeatCapacity_func new_PMI_MSC_HeatCapacity;
```

Optical Absorption

The optical absorption coefficient α in the photogeneration can be accessed through a PMI. For each optical beam, a model can be specified in the semabs section of the command file:

```
Physics {
    OptBeam (
        ...
        semabs (model = pmi_model_name)
        ...
    )
}
```

Dependencies

The optical absorption coefficient α may depend on the following variables:

energy	Optical wave energy E_{ph} [eV]
temperature	Temperature T [K]

The PMI model must compute the following result:

alpha	Optical absorption coefficient α [cm^{-1}]
-------	--

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Absorption : public PMI_Vertex_Interface {

public:
    PMI_Absorption (const PMI_Environment& env);
    virtual ~PMI_Absorption ();

    virtual void Compute_alpha
        (const double energy, const double t, double& alpha) = 0;
};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_Absorption* new_PMI_Absorption_func
    (const PMI_Environment& env);
extern "C" new_PMI_Absorption_func new_PMI_Absorption;
```

Example: Temperature-dependent Absorption Model

The following code computes a temperature-dependent value for the absorption coefficient α . This example is for illustration purposes only and does not pertain to any physical model:

```
#include "PMIModels.h"

class TempDep_Absorption : public PMI_Absorption {

private:
```

34: Physical Model Interface

Refractive Index

```
double Alpha;

public:
    TempDep_Absorption (const PMI_Environment& env);

    ~TempDep_Absorption () ;

    void Compute_alpha (const double energy, const double t, double& alpha);

};

TempDep_Absorption::TempDep_Absorption (const PMI_Environment& env) :
    PMI_Absorption (env)
{ Alpha = InitParameter ("Alpha", 1e5);
}

TempDep_Absorption::~TempDep_Absorption ()
{
}

void TempDep_Absorption::Compute_alpha (const double energy, const double t, double& alpha)
{ alpha = Alpha*(600 - t)/300;
}

extern "C"
PMI_Absorption* new_PMI_Absorption
    (const PMI_Environment& env)
{ return new TempDep_Absorption (env);
}
```

Refractive Index

The refractive index, n , in the photogeneration can be accessed through a PMI. For each optical beam, a model can be specified in the `RefractiveIndex` section of the command file:

```
Physics {
    RayTrace(
        ...
        RefractiveIndex(model = pmi_model_name)
        ...
    )
}
```

Dependencies

The refractive index, n , may depend on the following variables:

energy	Optical wave energy E_{ph} [eV]
temperature	Temperature T [K]

The PMI model must compute the following result:

Refract	Refractive index n (1)
---------	--------------------------

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_RefractiveIndex : public PMI_Vertex_Interface {  
  
public:  
    PMI_RefractiveIndex (const PMI_Environment& env);  
    virtual ~PMI_RefractiveIndex ();  
  
    virtual void Compute_Refra (const double energy, const double t, double&  
        Refra) = 0;  
};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_RefractiveIndex* new_PMI_RefractiveIndex_func  
    (const PMI_Environment& env);  
extern "C" new_PMI_RefractiveIndex_func new_PMI_RefractiveIndex;
```

Example: Temperature-dependent Refractive Index

The following code computes a simple temperature-dependent model for the refractive index:

```
#include "PMIModels.h"  
  
class TempDep_RefractiveIndex : public PMI_RefractiveIndex {  
  
private:  
    double RIndex;
```

34: Physical Model Interface

Stress

```
public:  
    TempDep_RefractiveIndex (const PMI_Environment& env);  
    ~TempDep_RefractiveIndex ();  
  
    void Compute_Refract (const double energy, const double temp, double& out);  
};  
  
TempDep_RefractiveIndex::  
TempDep_RefractiveIndex (const PMI_Environment& env) :  
    PMI_RefractiveIndex (env)  
{ RIndex = InitParameter ("RIndex", 42);  
}  
  
TempDep_RefractiveIndex::  
~TempDep_RefractiveIndex ()  
{  
}  
  
void TempDep_RefractiveIndex::  
Compute_Refract (const double energy, const double temp, double& Refract)  
{ Refract = 4*(1 + (temp-300)/100);  
}  
  
extern "C"  
PMI_RefractiveIndex* new_PMI_RefractiveIndex (const PMI_Environment& env)  
{ return new TempDep_RefractiveIndex (env);  
}
```

Stress

Sentaurus Device supports a PMI for mechanical stress (see [Chapter 23 on page 595](#)). The name of the PMI model must appear in the Piezo section of the command file:

```
Physics {  
    Piezo {  
        Stress = pmi_model_name  
    }  
}
```

Dependencies

A PMI stress model has no explicit dependencies. However, it can depend on doping concentrations and mole fractions through the run-time support.

The PMI model must compute the following results:

stress_xx	XX component of stress tensor [Pa]
stress_yy	YY component of stress tensor [Pa]
stress_zz	ZZ component of stress tensor [Pa]
stress_yz	YZ component of stress tensor [Pa]
stress_xz	XZ component of stress tensor [Pa]
stress_xy	XY component of stress tensor [Pa]

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Stress : public PMI_Vertex_Interface {  
  
public:  
    PMI_Stress (const PMI_Environment& env);  
    virtual ~PMI_Stress ();  
  
    virtual void Compute_StressXX  
        (double& stress_xx) = 0;  
  
    virtual void Compute_StressYY  
        (double& stress_yy) = 0;  
  
    virtual void Compute_StressZZ  
        (double& stress_zz) = 0;  
  
    virtual void Compute_StressYZ  
        (double& stress_yz) = 0;  
  
    virtual void Compute_StressXZ  
        (double& stress_xz) = 0;  
  
    virtual void Compute_StressXY
```

34: Physical Model Interface

Stress

```
    (double& stress_xy) = 0;  
};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_Stress* new_PMI_Stress_func  
    (const PMI_Environment& env);  
extern "C" new_PMI_Stress_func new_PMI_Stress;
```

Example: Constant Stress Model

The following code returns constant values for the stress tensor:

```
#include "PMIModels.h"  
  
class Constant_Stress : public PMI_Stress {  
  
private:  
    double xx, yy, zz, yz, xz, xy;  
  
public:  
    Constant_Stress (const PMI_Environment& env);  
  
    ~Constant_Stress ();  
  
    void Compute_StressXX (double& stress_xx);  
    void Compute_StressYY (double& stress_yy);  
    void Compute_StressZZ (double& stress_zz);  
    void Compute_StressYZ (double& stress_yz);  
    void Compute_StressXZ (double& stress_xz);  
    void Compute_StressXY (double& stress_xy);  
  
};  
  
Constant_Stress::  
Constant_Stress (const PMI_Environment& env) :  
    PMI_Stress (env)  
{  
    xx = InitParameter ("xx", 100);  
    yy = InitParameter ("yy", -4e9);  
    zz = InitParameter ("zz", 300);  
    yz = InitParameter ("yz", 400);  
    xz = InitParameter ("xz", 500);  
    xy = InitParameter ("xy", 600);  
}  
  
Constant_Stress::  
~Constant_Stress ()
```

```
{  
}  
  
void Constant_Stress::  
Compute_StressXX (double& stress_xx)  
{ stress_xx = xx;  
}  
  
void Constant_Stress::  
Compute_StressYY (double& stress_yy)  
{ stress_yy = yy;  
}  
  
void Constant_Stress::  
Compute_StressZZ (double& stress_zz)  
{ stress_zz = zz;  
}  
  
void Constant_Stress::  
Compute_StressYZ (double& stress_yz)  
{ stress_yz = yz;  
}  
  
void Constant_Stress::  
Compute_StressXZ (double& stress_xz)  
{ stress_xz = xz;  
}  
  
void Constant_Stress::  
Compute_StressXY (double& stress_xy)  
{ stress_xy = xy;  
}  
  
extern "C"  
PMI_Stress* new_PMI_Stress  
    (const PMI_Environment& env)  
{ return new Constant_Stress (env);  
}
```

34: Physical Model Interface

Trap Space Factor

Trap Space Factor

The space distribution of traps can be computed by a PMI (see [Chapter 10 on page 349](#)). The name of the PMI is specified in the Traps section as follows:

```
Physics {
    Traps (sFactor=pmi_model_name...)
}
```

NOTE The name of the PMI model must not coincide with the name of an internal field of Sentaurus Device. Otherwise, Sentaurus Device takes the value of the internal field as the space factor.

Dependencies

A PMI space factor model has no explicit dependencies. The model must compute:

spacefactor Space factor (1) or [cm⁻³]

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_SpaceFactor : public PMI_Vertex_Interface {

public:
    PMI_SpaceFactor (const PMI_Environment& env);
    virtual ~PMI_SpaceFactor () ;

    virtual void Compute_spacefactor
        (double& spacefactor) = 0;
};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_SpaceFactor* new_PMI_SpaceFactor_func
    (const PMI_Environment& env);
extern "C" new_PMI_SpaceFactor_func new_PMI_SpaceFactor;
```

Example: PMI User Field as Space Factor

The following code reads the space factor from a PMI user field:

```
#include "PMIModels.h"

class pmi_spacefactor : public PMI_SpaceFactor {

public:
    pmi_spacefactor (const PMI_Environment& env);
    ~pmi_spacefactor ();

    void Compute_spacefactor (double& spacefactor);
};

pmi_spacefactor::
pmi_spacefactor (const PMI_Environment& env) :
    PMI_SpaceFactor (env)
{
}

pmi_spacefactor::
~pmi_spacefactor ()
{
}

void pmi_spacefactor::
Compute_spacefactor (double& spacefactor)
{
    spacefactor = ReadUserField (PMI_UserField1);
}

extern "C"
PMI_SpaceFactor* new_PMI_SpaceFactor
    (const PMI_Environment& env)
{
    return new pmi_spacefactor (env);
}
```

Trap Capture and Emission Rates

The present PMI model can be used to define either capture and emission rates for traps (see c_C^n , c_V^p , e_C^n , and e_V^p in [Trap Occupation Dynamics on page 352](#)) or the transitions between states of multistate configurations (see [Specifying Multistate Configurations on page 367](#)). The model is an arbitrary function of n , p , T , T_n , T_p , and F .

Traps

In the command file, specify the model with the `CBRate` and `VBRate` options to `Traps`.

For example:

```
Traps ( (CBRate= ("modelX",17) VBRate="modelY") ... )
```

uses the user-specified model `modelX` to compute c_C^n and e_C^n (see [Local Trap Capture and Emission on page 354](#)), and `modelY` to compute c_V^p and e_V^p . The 17 is passed as the second argument to the virtual constructor for `modelX`; `modelY` is passed as a default value of 0 instead. The interpretation of these integers is left to the user-specified models. They allow you to select among different parameters or model variants, without having to reimplement the full model for each different choice of parameters or each minor model variation.

Multistate Configurations

The present model is used for transitions of multistate configurations by the keyword `CEModel` (see [Specifying Multistate Configurations on page 367](#)), for example:

```
MSCConfig { ...
    Transition ( Name="t1" To = "c" From="a" CEModel ("modelX" 5) )
}
```

where 5 is the optional model index parameter that defaults to 0.

Dependencies

The capture and emission rates may depend on the variables:

n	Electron density [cm ⁻³]
p	Hole density [cm ⁻³]

t	Lattice temperature [K]
tn	Electron temperature [K]
tp	Hole temperature [K]
f	Electric field [Vcm ⁻¹]

The PMI model must compute the following results (note that the carrier type that is captured and emitted is determined by the band to which the model is applied):

capture	Capture rate [s ⁻¹]
emission	Emission rate [s ⁻¹]
dcapturedn	Derivative of capture rate with respect to n [s ⁻¹ cm ³]
demissiondn	Derivative of emission rate with respect to n [s ⁻¹ cm ³]
dcapturedp	Derivative of capture rate with respect to p [s ⁻¹ cm ³]
demissiondp	Derivative of emission rate with respect to p [s ⁻¹ cm ³]
dcapturedt	Derivative of capture rate with respect to t [s ⁻¹ K ⁻¹]
demissiondt	Derivative of emission rate with respect to t [s ⁻¹ K ⁻¹]
dcapturedtn	Derivative of capture rate with respect to tn [s ⁻¹ K ⁻¹]
demissiondtn	Derivative of emission rate with respect to tn [s ⁻¹ K ⁻¹]
dcapturedtp	Derivative of capture rate with respect to tp [s ⁻¹ K ⁻¹]
demissiondtp	Derivative of emission rate with respect to tp [s ⁻¹ K ⁻¹]
dcapturedf	Derivative of capture rate with respect to f [s ⁻¹ V ⁻¹ cm]
demissiondf	Derivative of emission rate with respect to f [s ⁻¹ V ⁻¹ cm]

C++ Interface

The following base class is declared in `PMIModels.h`:

```
class PMI_TrapCaptureEmission : public PMI_Vertex_Interface {
public:
    PMI_TrapCaptureEmission(const PMI_Environment& env);
    virtual ~PMI_TrapCaptureEmission();

    virtual void Compute_rates
```

34: Physical Model Interface

Trap Capture and Emission Rates

```
(const double n, const double p, const double t,
    const double tn, const double tp, const double f,
    double& capture, double& emission) = 0;

virtual void Compute_dratesdn
    (const double n, const double p, const double t,
    const double tn, const double tp, const double f,
    double& dcapturedn, double& demissiondn) = 0;

virtual void Compute_dratesdp
    (const double n, const double p, const double t,
    const double tn, const double tp, const double f,
    double& dcapturedp, double& demissiondp) = 0;

virtual void Compute_dratesdt
    (const double n, const double p, const double t,
    const double tn, const double tp, const double f,
    double& dcapturedt, double& demissiondt) = 0;

virtual void Compute_dratesdtn
    (const double n, const double p, const double t,
    const double tn, const double tp, const double f,
    double& dcapturedtn, double& demissiondtn) = 0;

virtual void Compute_dratesdtp
    (const double n, const double p, const double t,
    const double tn, const double tp, const double f,
    double& dcapturedtp, double& demissiondtp) = 0;

virtual void Compute_dratesdf
    (const double n, const double p, const double t,
    const double tn, const double tp, const double f,
    double& dcapturedf, double& demissiondf) = 0;
};
```

The following virtual constructor must be implemented:

```
typedef PMI_TrapCaptureEmission* new_PMI_TrapCaptureEmission_func
    (const PMI_Environment& env, int id);
extern "C" new_PMI_TrapCaptureEmission_func new_PMI_TrapCaptureEmission;
```

Example: CEModel_ArrheniusLaw

The model has the structure of the Arrhenius law. It depends on the temperature.

Table 117 Model parameters

Symbol	Parameter name	Default	Unit	Description
δE	DeltaE	0.	eV	Energy difference
g	g	1.	1	Degeneracy factor
r_0	r0	1.	s ⁻¹	Maximal transition rate
E_{act}	Eact	0.	eV	Activation energy

Let δE and E_{act} be the energy difference and activation energy. The capture and emission rates are given by:

$$c = r_0 \exp(-E_{\text{act}}/kT) \quad (983)$$

$$e = r_0 g \exp(-\langle E_{\text{act}} + \delta E \rangle / kT) \quad (984)$$

This example is further discussed in [Example: Two-State Phase-Change Memory Model on page 371](#). The model is shipped with Sentaurus Device.

Trap Energy Shift

The present PMI determines a shift E_{shift} of trap energies that depends on the electric field and the lattice temperature at the location of the vertex or at a position given with ReferencePoint (see [Energetic and Spatial Distribution of Traps on page 350](#)).

Command File

The energy shift model is specified by `EnergyShift=<model name>` or `EnergyShift=(<model name>, <int>)` as an option to `Traps` in the `Physics` section. The optional integer defaults to zero and is passed as the argument `id` to the virtual constructor of the PMI (see below). The interpretation of `id` is dependent on the user-specified model.

34: Physical Model Interface

Trap Energy Shift

Dependencies

The trap energy shift can depend on the variables:

- f Electric field vector [Vcm⁻¹], a vector with up to three components, for the field in the x-, y-, and z-direction
- t Lattice temperature [K]

The PMI model must compute the following results:

- shift Energy shift [eV]
- dshiftdf Derivative of energy shift with respect to each component of f [eVcmV⁻¹], a vector with up to three entries
- dshiftdt Derivative of energy shift with respect to t [eVK⁻¹]

C++ Interface

The following base class is declared in `PMIModels.h`:

```
class PMI_TrapEnergyShift : public PMI_Vertex_Interface {
public:
    PMI_TrapEnergyShift(const PMI_Environment& env);
    virtual ~PMI_TrapEnergyShift();

    virtual void Compute_shift(
        const double f[3], const double t, double& shift) = 0;
    virtual void Compute_dshiftdf(
        const double f[3], const double t, double df[3]) = 0;
    virtual void Compute_dshiftdt(
        const double f[3], const double t, double& dt) = 0;
};
```

The following virtual constructor must be implemented:

```
typedef PMI_TrapEnergyShift* new_PMI_TrapEnergyShift_func
    (const PMI_Environment& env, int id);
extern "C" new_PMI_TrapEnergyShift_func new_PMI_TrapEnergyShift;
```

Piezoelectric Polarization

The effects of piezoelectric polarization can be modeled by adding the divergence of the piezoelectric polarization vector as an additional charge term:

$$q_{\text{PE}} = -\nabla \cdot P_{\text{PE}} \quad (985)$$

to the right-hand side of the Poisson equation (see [Eq. 26, p. 178](#)):

$$\nabla \cdot \epsilon \cdot \nabla \phi = -q(p - n + N_D - N_A + q_{\text{PE}}) \quad (986)$$

The quantity P_{PE} denotes the piezoelectric polarization vector, which may be defined by a PMI. The built-in models for piezoelectric polarization are discussed in [Dependency of Saturation Velocity on Stress on page 633](#).

The name of the PMI is specified in the `Physics` section of the command file as follows:

```
Physics {
    Piezoelectric_Polarization (pmi_polarization)
}
```

The piezoelectric polarization vector and the piezoelectric charge may be plotted by:

```
Plot {
    PE_Polarization/vector
    PE_Charge
}
```

Sentaurus Device assumes that the piezoelectric polarization vector P_{PE} is zero outside of the device. This boundary condition may lead to an unexpectedly large charge density if P_{PE} has a nonzero component orthogonal to the boundary (discontinuity in ∇P_{PE}).

Dependencies

The piezoelectric polarization model does not have explicit dependencies. However, it can use the run-time support. In particular, it has access to the stress fields.

The model must compute:

`pol` Piezoelectric polarization vector [Ccm^{-2}]

The resulting vector `pol` has the dimension 3. However, only the first `dim` components need to be defined, where `dim` is equal to the dimension of the problem.

34: Physical Model Interface

Piezoelectric Polarization

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Polarization : public PMI_Vertex_Interface {  
  
public:  
    PMI_Polarization (const PMI_Environment& env);  
    virtual ~PMI_Polarization ();  
  
    virtual void Compute_pol  
        (double pol [3]) = 0;  
};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_Polarization* new_PMI_Polarization_func  
    (const PMI_Environment& env);  
extern "C" new_PMI_Polarization_func new_PMI_Polarization;
```

Example: Gaussian Polarization Model

In this example, the piezoelectric polarization vector P_{PE} has a simple Gaussian shape in the x-direction:

```
#include "PMIModels.h"  
  
class Gauss_Polarization : public PMI_Polarization {  
  
private:  
    double x0, c, a;  
  
public:  
    Gauss_Polarization (const PMI_Environment& env);  
    ~Gauss_Polarization ();  
  
    void Compute_pol (double pol [3]);  
};  
  
Gauss_Polarization::  
Gauss_Polarization (const PMI_Environment& env) :  
    PMI_Polarization (env)  
{ x0 = InitParameter ("x0", 0.0);  
    c = InitParameter ("c", 1.0);
```

```

    a = InitParameter ("a", 1e-5);
}

Gauss_Polarization::
~Gauss_Polarization ()
{
}

void Gauss_Polarization::
Compute_pol (double pol [3])
{ double x, y, z;
  ReadCoordinate (x, y, z);
  pol [0] = a * exp (-c * (x-x0) * (x-x0));
  pol [1] = 0.0;
  pol [2] = 0.0;
}

extern "C"
PMI_Polarization* new_PMI_Polarization
  (const PMI_Environment& env)
{ return new Gauss_Polarization (env);
}

```

Incomplete Ionization

The ionization factors $G_D(T)$ and $G_A(T)$ (see [Incomplete Ionization Model on page 246](#)) can be defined by a PMI.

The name of the PMI should be specified in the `Physics` section of the command file as follows:

```

Physics {
  IncompleteIonization( Model( PMI_model_name("Species_name1
    Species_name2 ...") ) )
}

```

In addition, it is possible to have a PMI for each species separately:

```

Physics {
  IncompleteIonization(
    Model(
      PMI_model_name1("Species_name1")

```

34: Physical Model Interface

Incomplete Ionization

```
        PMI_model_name2 ("Species_name2")
    )
}
}
```

The species PMI parameters should be defined in the parameter file (see [Parameter File of Sentaurus Device on page 932](#)).

Dependencies

The ionization factors $G_D(T)$ and $G_A(T)$ may depend on the variable:

T Lattice temperature [K]

The PMI model must compute the following results:

g Ionization factor $G(T)$

dgdt Derivative of $G(T)$ with respect to T

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
enum PMI_SpeciesType {
    PMI_acceptor,
    PMI_donor
};

class PMI_DistributionFunction : public PMI_Vertex_Interface {

private:
    const PMI_SpeciesType speciesType;
    const char* speciesName;

public:
    PMI_DistributionFunction (const PMI_Environment& env,
                             const char* name
                             const PMI_SpeciesType type = PMI_acceptor);

    virtual ~PMI_DistributionFunction () ;

    PMI_SpeciesType SpeciesType () const { return speciesType; }
}
```

```

const char* SpeciesName () const { return speciesName; }

// read parameter from Sentaurus Device parameter file
// (override for PMI_Vertex_Interface::ReadParameter)
const PMIBaseParam* ReadParameter (const char* name) const;

// initialize parameter from Sentaurus Device parameter file or from default
value
// (override for PMI_Vertex_Interface::InitParameter)
double InitParameter (const char* name, double defaultvalue) const;

virtual void Compute_g
    (const double T,           // lattice temperature
     double& g) = 0;          // g = G(T)

virtual void Compute_dgdt
    (const double T,           // lattice temperature
     double& dgdt) = 0;        // dgdt = G' (T)

};

}

```

The prototype for the virtual constructor is given as:

```

typedef PMI_DistributionFunction* new_PMI_DistributionFunction_func
    (const PMI_Environment& env);
extern "C" new_PMI_DistributionFunction_func new_PMI_DistributionFunction;

```

Example: Matsuura Incomplete Ionization Model

The following C++ code implements the Matsuura model [1] for dopant Al in SiC material:

$$G_A(T) = 4 \exp\left(\frac{\Delta E_A - E_{\text{ex}}}{kT}\right) \cdot \left(g_1 + \sum_{r=2} \frac{g_r \exp\left(\frac{\Delta E_r - \Delta E_A}{kT}\right)}{r^2}\right) \quad (987)$$

where g_1 is the ground-state degeneracy factor, g_r is the $(r-1)$ -th excited state degeneracy factor, and ΔE_r is the difference in energy between the $(r-1)$ -th excited state level and E_V . ΔE_r is given by the hydrogenic dopant model [2]:

$$\Delta E_r = 13.6 \cdot \frac{m^*}{m_0 \cdot \varepsilon_s^2} \cdot \frac{1}{r^2} \quad [\text{eV}] \quad (988)$$

where m^* is the hole effective mass in SiC, and ε_s is the dielectric constant of SiC. The acceptor level is described as [2]:

$$\Delta E_A = \Delta E_1 + E_{\text{CCC}} \quad (989)$$

34: Physical Model Interface

Incomplete Ionization

where E_{CCC} is the energy induced due to central cell corrections.

The ensemble average E_{ex} of the ground and excited state levels of the acceptor is given by [3]:

$$E_{\text{ex}} = \frac{\sum_{r=2} (\Delta E_A - \Delta E_r) g_r \exp\left(-\frac{\Delta E_A - \Delta E_r}{kT}\right)}{\sum_{r=2} g_r \exp\left(-\frac{\Delta E_A - \Delta E_r}{kT}\right)} \quad (990)$$

The Matsuura model can be implemented as follows:

```
class Matsuura_DistributionFunction : public PMI_DistributionFunction {  
  
protected:  
    const double kB_300; // Boltzmann constant * 300 [eV]  
    int nb_item; // number of item in sum  
    double *gr, *dEr;  
    double Eex, dEA, Eccc;  
  
public:  
    Matsuura_DistributionFunction (const PMI_Environment& env,  
                                    const char* name,  
                                    const PMI_SpeciesType type = PMI_acceptor);  
  
    ~Matsuura_DistributionFunction ();  
  
    void Compute_g  
        (const double T, // lattice temperature  
         double& g); // g = G(T)  
  
    void Compute_dgdt  
        (const double T, // lattice temperature  
         double& dgdt); // dgdt = G'(T)  
  
    double Compute_Eex(double T); // compute Eex(T) Eq. 990, p. 1018  
    double Compute_dEexdT(double T); // compute dEex/dT  
  
};  
  
Matsuura_DistributionFunction::  
Matsuura_DistributionFunction (const PMI_Environment& env,  
                            const char* name,  
                            const PMI_SpeciesType type) :  
    PMI_DistributionFunction (env, name, type),  
    kB_300(1.380662e-23*300./1.602192e-19), // kB*T0/e0 = 0.02585199527 [eV]  
    Eex(0.)  
{
```

```

nb_item = InitParameter ("NumberOfItem", 1);
Eccc     = InitParameter ("Eccc", 0);

if(nb_item < 1) {
    printf("ERROR; PMI model Matsuura_DistributionFunction: parameter
NumberOfItem < 1 \n");
    exit(1);
}
gr = new double[nb_item];
dEr = new double[nb_item];

char str_r[6], name_gr[6];

int r;
for(r=0; r<nb_item; ++r) {
    name_gr[0] = 'g'; name_gr[1] = '\0';
    sprintf(str_r, "%d\0", r+1);
    strcat(name_gr, str_r);
    const PMIBaseParam* par = ReadParameter(name_gr);
    if(!par) {
        printf("ERROR; PMI model Matsuura_DistributionFunction: cannot read
parameter %s \n", name_gr);
        gr[r] = 2;
    } else
        gr[r] = *par;
}

const PMIBaseParam* par = ReadParameter("dE1");
if(!par) {
// dE[r] = 13.6 * m_eff/m0/eps/eps/r/r
    Compute_dEr(nb_item, dEr);
} else {
    char name_dEr[6];
    for(r=0; r<nb_item; ++r) {
        strcpy(name_dEr, "dE");
        sprintf(str_r, "%d\0", r+1);
        strcat(name_dEr, str_r);
        const PMIBaseParam* par = ReadParameter(name_dEr);
        if(!par) {
            printf("ERROR; PMI model Matsuura_DistributionFunction: cannot read
parameter %s \n", name_dEr);
            exit(1);
        }
        dEr[r] = *par;
    }
}
dEA = dEr[0] + Eccc;
}

```

34: Physical Model Interface

Incomplete Ionization

```
Matsuura_DistributionFunction::  
~Matsuura_DistributionFunction ()  
{  
    delete [] gr;  
    delete [] dEr;  
}  
  
void Matsuura_DistributionFunction::Compute_g  
    (const double T,           // lattice temperature  
     double& g)             // g = G(T)  
{  
    const double kT = kB_300*T/300;  
  
    Eex = Compute_Eex(T);  
    g = gr[0];  
  
    for(int r=1; r<nb_item; ++r) {  
        double delta = dEA - dEr[r];  
        g += gr[r]*exp( -delta/kT );  
    }  
    g *= 4.*exp( (dEA-Eex)/kT );  
}  
  
void Matsuura_DistributionFunction::Compute_dgdt  
    (const double T,           // lattice temperature  
     double& dgdt)          // dgdt = G' (T)  
{  
    const double kT = kB_300*T/300;  
  
    Eex = Compute_Eex(T);  
    double s1, s2 = gr[0], s3, s4 = 0., delta, tmp;  
  
    for(int r=1; r<nb_item; ++r) {  
        delta = dEA - dEr[r];  
        tmp   = gr[r]*exp( -delta/kT );  
        s2   += tmp;  
        s4   += tmp*delta/kT/T;           // s4 = ds2/dT  
    }  
  
    delta = dEA - Eex;  
    s1 = 4.*exp( delta/kT );  
    double dEEx_dT = Compute_dEExdT(T);  
    s3 = s1*( -dEEx_dT/kT - delta/kT/T );    // s3 = ds1/dT  
  
    dgdt = s3*s2 + s1*s4;           // dgdt = d(s1*s2)/dT  
  
    return;
```

```

}

// Eex is given by Eq. 990, p. 1018
double Matsuura_DistributionFunction::Compute_Eex(double T)
{
    const double kT = kB_300*T/300;

    double s1 = 0., s2 = gr[0];

    for(int r=1; r<nb_item; ++r) {
        double delta = dEA - dEr[r];
        double tmp   = gr[r]*exp( -delta/kT );
        s1 += delta*tmp;
        s2 += tmp;
    }

    return s1/s2;
}

double Matsuura_DistributionFunction::Compute_dEexdT(double T)
{
    const double kT = kB_300*T/300;

    double s1 = 0., s2 = gr[0], s3 = 0., s4 = 0.;

    for(int r=1; r<nb_item; ++r) {
        double delta = dEA - dEr[r];
        double tmp   = gr[r]*exp( -delta/kT );
        s1 += delta*tmp;
        s2 += tmp;
        s3 += delta*tmp*delta/kT/T; // s3 = ds1/dT
        s4 += tmp*delta/kT/T;      // s4 = ds2/dT
    }

    return (s3*s2 - s1*s4)/s2/s2;
}

extern "C"
PMI_DistributionFunction* new_PMI_DistributionFunction
(const PMI_Environment& env,
 const char* name,
 const PMI_SpeciesType type)
{
    return new Matsuura_DistributionFunction (env, name, type);
}

```

34: Physical Model Interface

Hot-Carrier Injection

```
void Compute_dEr(int nb_item, double* dEr)
{
    // dEr is given by Eq. 988, p. 1017

    // data from file: 6H-SiC.par
    const double epsilon = 9.66;                                // dielectric constant
    const double mh      = 1;                                    // hole effective mass in SiC

    const double E0 = 13.6*mh/epsilon/epsilon;

    for(int r=1; r<=nb_item; ++r) {
        dEr[r-1] = E0/r/r;
    }
}
```

Hot-Carrier Injection

Sentaurus Device provides a PMI for implementing hot-carrier injection models and computing the injection currents defined by the model. It is activated in the `Physics` interface section of the command file, `GateCurrent` subsection:

```
Physics(MaterialInterface="Silicon/Oxide"){
    GateCurrent( PMI_model(electron) )
}
```

The model can be used for both carrier types with either `PMI_model(electron)` or `PMI_model()`. The interface has access to the device mesh and device data (see [Vertex-based Run-Time Support for Multistate Configuration-dependent Models on page 922](#)).

Dependencies

A hot-carrier PMI model has no explicit dependencies. However, it can depend on any field at run-time using the access to device data.

The model must compute:

`gCurr` Vector of region/interface arrays with hot-carrier injection current densities; each region/interface array consists of the current density in each vertex of a region interface.

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_HotCarrierInjection : public PMI_Device_Interface {  
  
protected:  
    const PMI_CarrierType cType;  
  
public:  
    PMI_HotCarrierInjection (const PMI_Device_Environment& env,  
                           const PMI_CarrierType cType);  
    virtual ~PMI_HotCarrierInjection ();  
  
    virtual void Compute_gCurr  
        (const des_regioninterface_vector regioninterfaces,  
         // region interfaces associated with the model  
         des_array_vector& gCurr) = 0; // gate injection current in each vertex  
                                // of specified region interfaces  
};
```

Two virtual constructors are required for electron and hole hot injection:

```
typedef PMI_HotCarrierInjection* new_PMI_HotCarrierInjection_func  
    (const PMI_Device_Environment& env, const PMI_CarrierType carType);  
extern "C" new_PMI_HotCarrierInjection_func new_PMI_e_HotCarrierInjection;  
extern "C" new_PMI_HotCarrierInjection_func new_PMI_h_HotCarrierInjection;
```

Example: Lucky Model

The following example re-implements the built-in lucky injection model using the hot-carrier injection PMI:

```
#include "PMIModels.h"  
  
class PMI_LuckyModel : public PMI_HotCarrierInjection {  
  
private:  
    const des_mesh* mesh; //device mesh  
    des_data* data; //device data  
    const double*const* measure;  
    const double*const* surface_measure;  
  
public:  
    PMI_LuckyModel (const PMI_Device_Environment& env, const PMI_CarrierType
```

34: Physical Model Interface

Hot-Carrier Injection

```
carType);
~PMI_LuckyModel();

void Compute_gCurr(const des_regioninterface_vector regioninterfaces,
                   des_array_vector& gCurr);
};

void PMI_LuckyModel::Compute_gCurr(
const des_regioninterface_vector regioninterfaces, des_array_vector& gCurr)
{
    //compute current for each des_regioninterface associated with model
    for(int inter=0; inter < regioninterfaces.size(); inter++) {
        for(int k=0; k < regioninterfaces.at(inter)->size_vertex(); k++)
            gCurr[inter][k] = 0.0;

        des_regioninterface* ri = regioninterfaces.at(inter);
        des_bulk* b1 = ri->bulk1();
        des_bulk* b2 = ri->bulk2();

        //recognize regioninterface or regioninterface category
        if(((b1->material() == "Silicon") && (b2->material() == "Oxide")) ||
           ((b1->material() == "Oxide") && (b2->material() == "Silicon"))) {
            des_bulk* semReg = (b1->material() == "Silicon") ? b1 : b2;
            des_bulk* insReg = (b1->material() == "Silicon") ? b2 : b1;;
            //read DataEntries used in computation
            const double* pot = data->ReadScalar(des_data::vertex,
"ElectrostaticPotential");
            ...
            const double* eCurrent = (cType == PMI_Electron)
                ? data->ReadScalar(des_data::vertex, "eCurrentDensity") : NULL;

            //integrate over the corresponding semiconductor region
            for(size_t vi = 0; vi < semReg->size_vertex(); vi++) {
                des_vertex* rv = semReg->vertex(vi);
                //find the nearest interface vertex (distance to interface)
                des_vertex* NearestInterVertex;
                double NearestDistance;
                ...
                //find the nearest gate contact vertex (distance from NearestIntVertex
                to gate contact)
                des_vertex* NearestContVertex;
                double NearestContDistance;
                ...
                //coordinates and distances are in um, transform to cm
                double OxThickn = NearestContDistance*1.0e-4;
                double DistToSurf = NearestDistance*1.0e-4;

                //find the corresponding oxide element and its epsilon
            }
        }
    }
}
```

```

    ...
double OxConst = Epsilon[xEl->index()];
double OxImage0 = 1.6e-19/16/3.1452/8.85e-14/OxConst;

//for electrons
if(cType == PMI_Electron) {
    double eCur = 0.0;
    double eOxField = pot[NearestContVertex->index()]
        -pot[NearestInterVertex->index()];
    eOxField = eOxField > 0 ? OxField[NearestInterVertex->index()]
        : -OxField[NearestInterVertex->index()];

    double barrier;
    if(pot[NearestInterVertex->index()] < pot[NearestContVertex-
>index()])
        barrier = eBarrierHeight-eAlfa*sqrt(fabs(eOxField)) -
eBeta*pow(fabs(eOxField),2.0/3.0);
    else
        barrier = eBarrierHeight+(pot[NearestInterVertex->index()]
            -pot[NearestContVertex->index()]) -
eAlfa*sqrt(fabs(eOxField))
            -eBeta*pow(fabs(eOxField),2.0/3.0);
    if(barrier < 0) barrier = 0.0;
    double eBarrierLoc = barrier;
    double p1 = 0.0;
    double eEnergy = eField[rv->index()]*eLambd;
    if(eEnergy > 1.0e-30) {
        p1 = 0.25;
        if (eBarrierLoc > 1.0e-30) p1 = 0.25*eEnergy/eBarrierLoc*exp(-
eBarrierLoc/eEnergy);
    }
    double p2 = exp(-DistToSurf/eLambd);
    double eDistFromSurf = 1.0e30;
    if (eOxField > 1.0e-30) eDistFromSurf = sqrt(OxImage0/eOxField);
    double p3 = exp(-eDistFromSurf/eOxLambd);

    //compute the node measure
    double node_measure;

    eCur = eCurrent[rv->index()]*p1*p2*p3*node_measure/eLambdR;
    gCurr[inter][NearestInterVertexRIind] += eCur/risurface;
}

//for holes
if(cType == PMI_Hole) {
    ...
}
}//end integration on semiconductor region

```

34: Physical Model Interface

Piezoresistive Coefficients

```
    } //end model implementation
} //end loop over regioninterfaces associated with "PMI_HotCarrierInjection"
model
}
extern "C" {
    PMI_HotCarrierInjection* new_PMI_e_HotCarrierInjection
        (const PMI_Device_Environment& env, const PMI_CarrierType carType)
    {
        return new PMI_LuckyModel(env, carType);
    }
}
extern "C" {
    PMI_HotCarrierInjection* new_PMI_h_HotCarrierInjection
        (const PMI_Device_Environment& env, const PMI_CarrierType carType)
    {
        return new PMI_LuckyModel(env, carType);
    }
}
```

Piezoresistive Coefficients

Sentaurus Device provides a PMI for implementing the dependencies of the piezoresistive prefactors over the normal electric field (see [Enormal- and MoleFraction-dependent Piezo Coefficients on page 609](#)). It is activated in the Piezo section of the command file, in the Tensor subsection:

```
Physics {
    ...
    Piezo(
        Model(Mobility(Tensor("pmi_model")))
    )
}
```

Dependencies

The piezoresistive prefactors e_{Pij} and h_{Pij} may depend on the following variable:

Enormal	Normal to interface semiconductor-dielectric electric field [Vcm ⁻¹]
---------	--

The PMI model must compute the following results:

p11, p12, p44	Piezoresistive prefactors [1]
dp11, dp12, dp44	Derivatives of p_{ij} with respect to E_{normal} [$V^{-1}cm$]

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_PiezoresistanceFactor : public PMI_Vertex_Interface
{
public:
    PMI_PiezoresistanceFactor (const PMI_Environment& env);
    virtual ~PMI_PiezoresistanceFactor () ;

    // methods to be implemented by user
    virtual void Compute_Pij(const double Enormal,
        double& p11, double& p12, double& p44) = 0;

    virtual void Compute_DerPij(const double Enormal,
        double& dp11, double& dp12, double& dp44) = 0;
};
```

Two virtual constructors are required for the calculation of the piezoresistive prefactors.

```
typedef PMI_PiezoresistanceFactor* new_PMI_PiezoresistanceFactor_func
    (const PMI_Environment& env);
extern "C" new_PMI_PiezoresistanceFactor_func new_PMI_ePiezoresistanceFactor;
extern "C" new_PMI_PiezoresistanceFactor_func new_PMI_hPiezoresistanceFactor;
```

Current Plot File of Sentaurus Device

The current plot PMI allows user-computed entries to be added to the current plot file. It is specified in the `CurrentPlot` section of the command file, for example:

```
CurrentPlot {
    pmi_CurrentPlot
}
```

The interface has access to the device mesh and device data (see [Vertex-based Run-Time Support for Multistate Configuration-dependent Models on page 922](#)).

Structure of Current Plot File

A current plot file consists of a header section and a data section. For each function, the structure can be described as follows:

```
dataset name
function name
value0
value1
...
```

A dataset name denotes a dataset, for example:

```
time
Tmin
```

If a dataset corresponds to a region or contact, it is customary to add the region or contact name:

```
gate Charge
```

The function name describes the function, for example:

```
ElectrostaticPotential
Temperature
```

Afterwards, a function value is added to the current plot file for each plot time point.

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_CurrentPlot : public PMI_Device_Interface {

public:
    PMI_CurrentPlot (const PMI_Device_Environment& env);
    virtual ~PMI_CurrentPlot () ;

    virtual void Compute_Dataset_Names
        (des_string_vector& dataset) = 0;

    virtual void Compute_Function_Names
        (des_string_vector& function) = 0;

    virtual void Compute_Plot_Values
        (des_double_vector& value) = 0;
};
```

The methods `Compute_Dataset_Names()` and `Compute_Function_Names()` are used to generate the header in the current plot file (see [Structure of Current Plot File on page 1028](#)). `Compute_Plot_Values()` is called for each plot time point to compute the plot values. Use the `push_back()` function to add values to the arrays `dataset`, `function`, or `value`.

NOTE All three methods `Compute_Dataset_Names()`, `Compute_Function_Names()`, and `Compute_Plot_Values()` must always compute the same number of values. Otherwise, an inconsistent current plot file will be generated.

The prototype for the virtual constructor is given as:

```
typedef PMI_CurrentPlot* new_PMI_CurrentPlot_func
(const PMI_Device_Environment& env);
extern "C" new_PMI_CurrentPlot_func new_PMI_CurrentPlot;
```

Example: Average Electrostatic Potential

The following example computes regionwise averages for the electrostatic potential. This is the same functionality as provided by the built-in current plot command (see [CurrentPlot Section on page 59](#)):

```
class CurrentPlot : public PMI_CurrentPlot {
private:
    typedef std::vector<des_bulk*> des_bulk_vector;

    const des_mesh* mesh;      // device mesh
    des_bulk_vector regions; // list of semiconductor bulk regions
    double scale;            // scaling factor

public:
    CurrentPlot (const PMI_Device_Environment& env);
    ~CurrentPlot ();

    void Compute_Dataset_Names (des_string_vector& dataset);
    void Compute_Function_Names (des_string_vector& function);
    void Compute_Plot_Values (des_double_vector& value);
};

CurrentPlot::
CurrentPlot (const PMI_Device_Environment& env) :
    PMI_CurrentPlot (env)
{ mesh = Mesh ();
    // determine regions to process
    for (size_t ri = 0; ri < mesh->size_region (); ri++) {
```

34: Physical Model Interface

Current Plot File of Sentaurus Device

```
des_region* r = mesh->region (ri);
if (r->type () == des_region::bulk) {
    des_bulk* b = dynamic_cast <des_bulk*> (r);
    if (b->material () != "Oxide") {
        // we found a semiconductor bulk region
        regions.push_back (b);
    }
}
}

// read parameters
scale = InitParameter ("scale", 0.0);
}

CurrentPlot::
~CurrentPlot ()
{
}

void CurrentPlot::
Compute_Dataset_Names (des_string_vector& dataset)
{ for (size_t ri = 0; ri < regions.size (); ri++) {
    des_bulk* b = regions [ri];
    std::string name = "Average_";
    name += b->name ();
    name += "ElectrostaticPotential";
    dataset.push_back (name);
}
}

void CurrentPlot::
Compute_Function_Names (des_string_vector& function)
{ for (size_t ri = 0; ri < regions.size (); ri++) {
    function.push_back ("ElectrostaticPotential");
}
}

void CurrentPlot::
Compute_Plot_Values (des_double_vector& value)
{ des_data* data = Data ();
    const double*const* measure = data->ReadMeasure ();
    const double* pot = data->ReadScalar (des_data::vertex,
"ElectrostaticPotential");
    for (size_t ri = 0; ri < regions.size (); ri++) {
        des_bulk* b = regions [ri];

        double sum_pot = 0.0;
        double sum_measure = 0.0;
```

```

        for (size_t ei = 0; ei < b->size_element (); ei++) {
            des_element* e = b->element (ei);
            for (size_t vi = 0; vi < e->size_vertex (); vi++) {
                des_vertex* v = e->vertex (vi);
                const double m = measure [e->index ()][vi];
                const double p = pot [v->index ()];
                sum_pot += m * p;
                sum_measure += m;
            }
        }
        value.push_back (scale * (sum_pot / sum_measure));
    }
}

extern "C" {
PMI_CurrentPlot* new_PMI_CurrentPlot (const PMI_Device_Environment& env)
{
    return new CurrentPlot (env);
}
}

```

Postprocess for Transient Simulation

The postprocess PMI allows you to post-compute data during a transient simulation. The PMI is called after a transient time step has succeeded. It is specified in the Math section of the command file, for example:

```

Math {
    PostProcess (
        Transient = "pmi_postprocess"
    )
}

```

The interface provides access to the device mesh and device data (see [Vertex-based Run-Time Support for Multistate Configuration–dependent Models on page 922](#)).

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_PostProcess : public PMI_Device_Interface {  
  
public:  
    PMI_PostProcess (const PMI_Device_Environment& env);  
    virtual ~PMI_PostProcess ();  
  
    virtual void Compute_PostProcess () = 0;  
};
```

The method `Compute_PostProcess()` is called after the transient time step has successfully completed.

Example: Postprocess User-Field

The following code modifies the user-field depending on the current temperature change and transient step size after the transient time step has succeeded:

```
#include "PMIModels.h"  
  
class PostProcess : public PMI_PostProcess {  
  
private:  
    const des_mesh* mesh;  
  
public:  
    PostProcess (const PMI_Device_Environment& env);  
    ~PostProcess ();  
    void Compute_PostProcess ();  
};  
  
PostProcess::  
PostProcess (const PMI_Device_Environment& env) :  
    PMI_PostProcess (env)  
{  
    mesh = Mesh();  
}  
  
PostProcess::  
~PostProcess ()  
{  
}
```

```

void PostProcess::
Compute_PostProcess ()
{
    des_data* data = Data();

    const double* T = data->ReadScalar(des_data::vertex,
        "LatticeTemperature");
    const double* T0 = data->ReadScalar(des_data::vertex, "PMIUserField0");

    double* delta_T = new double [mesh->size_vertex()];
    for (int vi=0; vi<mesh->size_vertex(); vi++) {
        delta_T[vi] = (T[vi]-T0[vi])/ReadTransientStepSize();
    }
    data->WriteScalar(des_data::vertex, "PMIUserField0", T);
    data->WriteScalar(des_data::vertex, "PMIUserField1", delta_T);
    if (delta_T!=NULL) { delete [] delta_T; }
}

```

Special Contact PMI for Raytracing

The PMI for raytracing allows you to access and change the parameters of a ray at special contacts. These contacts are drawn in the same manner as electrodes and thermodes (see [Boundary Condition for Raytracing on page 422](#)). The raytrace PMI is specified in the RayTraceBC section of the command file:

```

RayTraceBC {...  
    { Name = "pmi_contact"  
        PMIModel = "pmi_modelname"  
    }  
}

```

The name of "pmi_contact" must match the contact name in the device. Any ray that hits this special contact invokes a call to this PMI. This raytrace PMI works only with the raytracer (see [Raytracer on page 407](#)) and the complex refractive index model (see [Complex Refractive Index Model on page 477](#)).

NOTE When using multithreading of the raytracer, you must carefully design the PMI code to be thread safe. This means that global variables should not be used since multiple threads may change the global variables at the same time, leading to confusion in the user PMI code at run-time.

Dependencies

The raytrace PMI depends on the following variables:

wavelength	Wavelength [cm]
incident_angle	Incident angle [radian]
*incident_dirvec	Direction vector of incident ray
*polarvec	Polarization vector of incident ray
*normalvec	Normal vector to surface of impingement
*intersectpoint	Intersection position vector [cm]
*region1_name	Name of region 1 (string)
*region2_name	Name of region 2 (string)
n1_real	Real part of refractive index 1
n1_imag	Imaginary part of refractive index 1
n2_real	Real part of refractive index 2
n2_imag	Imaginary part of refractive index 2
reflected_angle	Reflected angle [radian]
transmitted_angle	Transmitted angle [radian]
*reflected_dirvec	Direction vector of reflected ray
*transmitted_dirvec	Direction vector of transmitted ray
*reflected_startposition	Starting position vector of reflected ray [cm]
*transmitted_startposition	Starting position vector of transmitted ray [cm]
R_TE	Power TE reflection coefficient
T_TE	Power TE transmission coefficient
R_TM	Power TM reflection coefficient
T_TM	Power TM transmission coefficient

To obtain the rate intensity (units of s^{-1}) carried by the ray, you need to compute the square of the length of *polarvec. This rate intensity includes all previous absorptions sustained by the ray when it traversed absorptive regions of the device, and it can be used directly to compute

the amount of optical generation (with units of s^{-1}). However, for raytracing used in LED simulations, the square of the length of *polarvec gives only a relative intensity value because the polarization vector of the head ray of the raytree is initialized to a length of 1.0.

If the reflection or transmission coefficients are equal to zero, no reflected or transmitted rays are created in the raytracing process.

With this PMI model, you can change the following variables:

reflected_angle	Reflected angle [radian]
transmitted_angle	Transmitted angle [radian]
*reflected_dirvec	Direction vector of reflected ray
*transmitted_dirvec	Direction vector of transmitted ray
*reflected_startposition	Starting position vector of reflected ray [cm]
*transmitted_startposition	Starting position vector of transmitted ray [cm]
R_TE	Power TE reflection coefficient
T_TE	Power TE transmission coefficient
R_TM	Power TM reflection coefficient
T_TM	Power TM transmission coefficient

If you want to change the direction vector, angle, or position vector of the reflected or transmitted ray, the respective flags must be set to TRUE:

- is_reflectedangle_changed
- is_reflecteddirvec_changed
- is_transmittedangle_changed
- is_transmitteddirvec_changed
- is_reflected_new_startposition
- is_transmitted_new_startposition

However, no flags are needed if you want to change the power reflection or transmission coefficients.

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_EXTERNAL PMI_RayTraceBoundary : public PMI_Vertex_Interface {
public:
    PMI_RayTraceBoundary (const PMI_Environment& env);
    virtual ~PMI_RayTraceBoundary();

    // methods to be implemented by user
    virtual void Compute_BoundaryParameters
        // Non-changeable quantities
        const double wavelength,                                // wavelength [cm]
        const double incident_angle,                            // incident angle
        const double* incident_dirvec,                         // dir vector of incident ray
        const double* polarvec,                               // polar vec of incident ray
        const double* normalvec,                             // normal to impingement
        const double* intersectpoint,                        // intersection point
        const char* region1_name,                           // name of region 1
        const char* region2_name,                           // name of region 2
        const double n1_real,                               // real part of refr index 1
        const double n1_imag,                               // imag part of refr index 1
        const double n2_real,                               // real part of refr index 2
        const double n2_imag,                               // imag part of refr index 2
        // User changeable quantities
        bool& is_reflectedangle_changed,                  // is refl angle changed?
        bool& is_reflecteddirvec_changed,                // is reflected dir changed?
        bool& is_transmitedangle_changed,                 // is transm angle changed?
        bool& is_transmiteddirvec_changed,               // is transm dir changed?
        bool& is_reflected_new_startposition,            // is refl pos changed?
        bool& is_transmited_new_startposition,           // is transm pos changed?
        double& reflected_angle,                          // reflected angle
        double& transmitted_angle,                        // transmitted angle
        double* reflected_dirvec,                         // dir vec of reflected ray
        double* transmitted_dirvec,                       // dir vec of transmitted ray
        double* reflected_startposition,                 // start pos of reflected ray
        double* transmitted_startposition,               // start pos of transm ray
        double& R_TE,                                   // power TE reflection coeff.
        double& T_TE,                                   // power TE transmission coeff.
        double& R_TM,                                   // power TM reflection coeff.
        double& T_TM,                                   // power TM transmission coeff.
    ) = 0;

    // Auxiliary functions for users
    void ReadComplexRefractiveIndex(std::string materialname,
        double wavelength,      // in microns
        double& n,
```

```
    double& k);
private:
    const PMI_Environment* thisenv;
};
```

An internal auxiliary function called `ReadComplexRefractiveIndex(...)` has been implemented to allow you to compute the complex refractive index of any material at a desired wavelength. The constraint is that the material must be defined in the parameter file.

Example: Assessing and Modifying a Ray

The following example shows how to access the information about a ray that intersects the special raytrace PMI contact, and how you can change the information of the ray:

```
class Dummy_RayTraceBoundary : public PMI_RayTraceBoundary {
private:
    // short ind_field; // field index for optical generation
    //     double shape; // transient curve shape
    //     double G1, G2, G3, T1, T2, T3, T4; // const for shapes
    int count;
    double d1;
public:
    Dummy_RayTraceBoundary (const PMI_Environment& env);
    ~Dummy_RayTraceBoundary () ;

    void Compute_BoundaryParameters
        (const double wavelength,                      // wavelength [cm]
         const double incident_angle,                  // incident angle
         const double* incident_dirvec,                // dir vec of incident ray
         const double* polarvec,                      // polar vec of incident ray
         const double* normalvec,                     // normal to impingement
         const double* intersectpoint,                // intersection point
         const char* region1_name,                    // name of region 1
         const char* region2_name,                    // name of region 2
         const double n1_real,                        // real part of refr index 1
         const double n1_imag,                        // imag part of refr index 1
         const double n2_real,                        // real part of refr index 2
         const double n2_imag,                        // imag part of refr index 2
         // User changeable quantities
         bool& is_reflectedangle_changed,           // is refl angle changed?
         bool& is_reflecteddirvec_changed,          // is refl dir changed?
         bool& is_transmittedangle_changed,          // is transm angle changed?
         bool& is_transmitteddirvec_changed,         // is transm dir changed?
         bool& is_reflected_new_startposition,       // is refl pos changed?
         bool& is_transmitted_new_startposition,     // is transm pos changed?
         double& reflected_angle,                  // reflected angle
```

34: Physical Model Interface

Special Contact PMI for Raytracing

```
        double& transmitted_angle,                                // transmitted angle
        double* reflected_dirvec,                             // dir vec of reflected ray
        double* transmitted_dirvec,                           // dir vec of transmitted ray
        double* reflected_startposition,                      // start pos of reflected ray
        double* transmitted_startposition,                   // start pos of transm ray
        double& R_TE,                                       // power TE reflection coeff.
        double& T_TE,                                       // power TE transmission coeff.
        double& R_TM,                                       // power TM reflection coeff.
        double& T_TM                                         // power TM transmission coeff.
    );
}

Dummy_RayTraceBoundary::  
Dummy_RayTraceBoundary (const PMI_Environment& env) :  
    PMI_RayTraceBoundary (env)  
{  
    printf("PMI: initializing ray trace PMI\n");  
}

Dummy_RayTraceBoundary::  
~Dummy_RayTraceBoundary ()  
{  
}

void Dummy_RayTraceBoundary::  
Compute_BoundaryParameters (  
    const double wavelength,  
    const double incident_angle,  
    const double* incident_dirvec,  
    const double* polarvec,  
    const double* normalvec,  
    const double* intersectpoint,  
    const char* region1_name,  
    const char* region2_name,  
    const double n1_real,  
    const double n1_imag,  
    const double n2_real,  
    const double n2_imag,  
    // User changeable quantities  
    bool& is_reflectedangle_changed,  
    bool& is_reflecteddirvec_changed,  
    bool& is_transmittedangle_changed,  
    bool& is_transmitteddirvec_changed,  
    bool& is_reflected_new_startposition,  
    bool& is_transmitted_new_startposition,  
    double& reflected_angle,  
    double& transmitted_angle,  
    double* reflected_dirvec,
```

```

        double* transmitted_dirvec,
        double* reflected_startposition,
        double* transmitted_startposition,
        double& R_TE,
        double& T_TE,
        double& R_TM,
        double& T_TM
    )
{
    // Ray goes from region 1 to region 2.
    // PMI contact is the interface between regions 1 and 2.
    printf("Region 1: Name=%s, Refractive Index = %e + i%e\n",
        region1_name, n1_real, n1_imag);
    printf("Angles: Incident=%lf, Reflected=%lf, Transmitted=%lf\n",
        incident_angle, reflected_angle, transmitted_angle);
    printf("Wavelength = %e [cm]\n", wavelength);
    printf("Incident Ray Direction=(%e,%e,%e)\n",
        incident_dirvec[0], incident_dirvec[1], incident_dirvec[2]);
    printf("Power Coefficients: R_TE=%e, T_TE=%e, R_TM=%e, T_TM=%e\n",
        R_TE, T_TE, R_TM, T_TM);

    // For example, change reflected direction to (1,2,3)
    is_reflecteddirvec_changed = TRUE;
    reflected_dirvec[0] = 1.0;
    reflected_dirvec[1] = 2.0;
    reflected_dirvec[2] = 3.0;
}

extern "C" {
    PMI_RayTraceBoundary* new_PMI_RayTraceBoundary (const PMI_Environment& env)
    { return new Dummy_RayTraceBoundary (env);
    }
}
}

```

Spatial Distribution Function

The spatial distribution function $R(w, l, E)$ can be defined by a PMI (see [Heavy Ions on page 492](#)).

The name of the PMI must be specified in the Physics section of the command file as follows:

```

Physics {
    HeavyIon(
        SpatialShape = PMI_shape_name
    )
}

```

34: Physical Model Interface

Spatial Distribution Function

```
}
```

or:

```
Physics {
    HeavyIon(
        SpatialShape = PMI_shape_name(Energy = value) # [eV]
    )
}
```

Dependencies

The spatial distribution function $R(w, l, E)$ depends on the following variables:

- w Radius defined as the perpendicular distance from the track [cm]
- l Coordinate along the track [cm]
- E Energy of heavy ion [eV]

The PMI model must compute the following result:

- R The value of the spatial distribution $R(w, l, E)$ [1]

C++ Interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_SpatialDistributionFunction: public PMI_Vertex_Interface {
    // * The spatial distribution function:
    //
    // * R(w,l,E)
    //
    // * where
    // * - l is the coordinate along the particle path [um];
    // * - w is radial coordinate orthogonal to l [um];
    // * - E is energy of heavy ion [eV];

    public:
        PMI_SpatialDistributionFunction (const PMI_Environment& env, const char*
        IonType);

        virtual ~PMI_SpatialDistributionFunction () ;
```

```
// user-defined name of heavy ion (see Using Alpha Particle Model on page 490)
const char* GetHeavyIonType () const;

// methods to be implemented by user
virtual void Compute_R(double& R, const double w, const double l = -1., const
double E = -1.) = 0;

};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_SpatialDistributionFunction*
new_PMI_SpatialDistributionFunction_func
(const PMI_Environment& env, const char* HeavyIonName);
new_PMI_SpatialDistributionFunction_func new_PMI_SpatialDistributionFunction;
```

Example: Gaussian Spatial Distribution Function

The built-in Gaussian spatial distribution function (see [Heavy Ions on page 492](#)) can also be implemented as a PMI model:

```
#include "PMIModels.h"

class Gaussian_SpatialDistributionFunction : public
PMI_SpatialDistributionFunction {

public:
    Gaussian_SpatialDistributionFunction (const PMI_Environment& env, const
char* HeavyIonName);
    ~Gaussian_SpatialDistributionFunction ();

    void Compute_R
        (double& R, const double w, const double l, const double E);

};

Gaussian_SpatialDistributionFunction::
Gaussian_SpatialDistributionFunction (const PMI_Environment& env, const char*
HeavyIonName) :
    PMI_SpatialDistributionFunction (env, HeavyIonName)
{ }

Gaussian_SpatialDistributionFunction::
~Gaussian_SpatialDistributionFunction ()
{ }
```

34: Physical Model Interface

References

```
void Gaussian_SpatialDistributionFunction::  
Compute_R(double& R, const double w, const double l, const double E)  
{  
    //      the unit w,l is [cm]  
    //      the unit E is [eV] (in this implementation not used)  
    //      R(w,l,E) = exp( -(w/wt(l))^2 )  
  
    double wt = 1.e-4; // scaling factor 1um = 1.e-4cm  
    double x = w/wt;  
  
    R = exp(-x*x);  
}  
  
extern "C"  
PMI_SpatialDistributionFunction* new_PMI_SpatialDistributionFunction  
(const PMI_Environment& env, const char* HeavyIonName)  
{  
    return new Gaussian_SpatialDistributionFunction (env, HeavyIonName);  
}
```

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.

Part VI Appendices

This part of the Sentaurus Device manual contains the following appendices:

[Appendix A Mathematical Symbols on page 1045](#)

[Appendix B Syntax on page 1049](#)

[Appendix C File-naming Convention on page 1051](#)

[Appendix D Command-Line Options on page 1053](#)

[Appendix E Run-Time Statistics on page 1057](#)

[Appendix F Data and Plot Names on page 1059](#)

[Appendix G Command File Overview on page 1087](#)

Mathematical Symbols

This appendix contains notational conventions and a list of symbols used in the Sentaurus Device User Guide.

They are listed alphabetically. Non-Latin characters are sorted according to their English translation.

/	Division, right binding: $a/bc = a/(bc)$
\hat{a}	Unit (3D) vector
\vec{a}	(3D) vector
$ a $	Absolute value of a scalar
$ \vec{a} $	Absolute value of a vector $a = \vec{a} $
a^T	Transpose of a vector or a matrix
a^{-1}	Inverse of function or matrix, reciprocal of a scalar
$\vec{a} \cdot \vec{b}$	Inner (dot) product
$\vec{a} \times \vec{b}$	Vector (cross) product
$\vec{a}\vec{b}$	Dyadic product: $(\vec{a}\vec{b})_{ij} = \vec{a}_i\vec{b}_j$
$\langle ab\bar{c} \rangle$	Miller indices. a , b , and c are digits; a bar over a digit indicates negation applied to that particular digit.
<hr/>	
χ	Electron affinity or thermal resistivity
c_L	Lattice heat capacity
e	Base of natural logarithm
\vec{E}	Electric field
E_{bgn}	Bandgap narrowing
E_C	Conduction band energy

A: Mathematical Symbols

$E_{F,n}$	Electron quasi-Fermi energy
$E_{F,p}$	Hole quasi-Fermi energy
E_g	Intrinsic band gap
$E_{g,eff}$	Effective band gap, $E_{g,eff} = E_g - E_{bgn}$
E_v	Valence band energy
ϵ	Absolute dielectric constant of a material
ϵ_0	Dielectric constant of vacuum
\vec{F}	Electric field
F_α	Integral of distribution function; for Fermi statistics, Fermi integral of order α
G	Generation rate (does not include recombination)
γ_n	Degeneracy factor for electrons
γ_p	Degeneracy factor for holes
\hbar	Planck's constant divided by 2π
i	Imaginary unit, $i^2 = -1$
Im	Imaginary part
\vec{J}_D	Displacement current density
\vec{J}_M	Current density in metals
\vec{J}_n	Electron current density
\vec{J}_p	Hole current density
k	Boltzmann constant
κ_L	Lattice thermal conductivity
κ_n	Electron thermal conductivity
κ_p	Hole thermal conductivity

Λ_n	Electron quantum potential
Λ_p	Hole quantum potential
\ln	Natural logarithm
m_0	Free electron mass
m_n	Electron density-of-states mass
m_p	Hole density-of-states mass
μ_n	Electron mobility
μ_p	Hole mobility
n	Electron density
\hat{n}	Unit normal vector
$N_{A, 0}$	Chemically active acceptor concentration
N_A	Ionized acceptor concentration
N_C	Conduction band density-of-states
$N_{D, 0}$	Chemically active donor concentration
N_D	Ionized donor concentration
n_i	Intrinsic density (not accounting for bandgap narrowing)
$n_{i, \text{eff}}$	Effective intrinsic density (accounting for bandgap narrowing)
N_i	Ionized dopant concentration, $N_i = N_A + N_D$
n_{se}	Single exciton density
N_{tot}	Total doping concentration, $N_{\text{tot}} = N_{A, 0} + N_{D, 0}$
N_V	Valence band density-of-states
p	Hole density
\vec{P}	Polarization
P_n	Electron thermoelectric power

A: Mathematical Symbols

P_p	Hole thermoelectric power
Φ_M	Metal Fermi potential
Φ_n	Electron quasi-Fermi potential
Φ_p	Hole quasi-Fermi potential
ϕ	Electrostatic potential
q	Elementary charge
r	Anisotropy factor
R	Recombination rate (does not include generation)
R_{net}	Net recombination rate, $R_{\text{net}} = R - G$
Re	Real part
\vec{S}_L	Lattice heat flux density
\vec{S}_n	Electron heat flux density
\vec{S}_p	Hole heat flux density
T	Lattice temperature
T_n	Electron temperature
T_p	Hole temperature
τ_n	Electron lifetime
τ_p	Hole lifetime
Θ	Unit step function (0 for negative arguments, 1 for positive arguments)
\vec{v}_n	Electron drift velocity
\vec{v}_p	Hole drift velocity
$\vec{v}_{\text{sat},n}$	Electron saturation velocity
$\vec{v}_{\text{sat},p}$	Hole saturation velocity

APPENDIX B Syntax

The syntax of the input file of Sentaurus Device, and the basic syntactical and lexical conventions are described here.

Sentaurus Device has a hierarchical input syntax. At the lowest level, device, system, and solve information is specified as well as the default and global parameters. Inside each Device section, the parameters specific to one device type can be specified.

Inside the System section, the real devices are specified or ‘instantiated.’ Here, parameters can be given that are specific to one instantiation of a device. The input file is a collection of specifications used to establish the simulation environment with actions describing which equations must be solved and how they must be solved. The syntax of the input file contains several entry types. All basic input file entries adhere to the syntactical and lexical rules described in [Table 118](#).

Table 118 Entry types in Sentaurus Device

Entry type	Description
Keyword	These are the known names of the input file. They are case insensitive. Therefore, the following keywords are all equivalent: Quasistationary, QuasiStationary, and quasistationary. Most keywords can be abbreviated. The above example can also be written as QuasiStat.
Integer	These are (possibly) signed decimal numbers. The following integers are valid: 123, -73492, 0.
Float	Floating point numbers are compatible with the C language format for floating point numbers. The following floating point numbers are valid: 123, 123.0, 1.23e2, -1.23E2.
Vector	Vectors in real space are defined depending on the actual dimension. In 3D, a vector is specified by three floating point numbers; in 2D, by two floating point numbers enclosed in parentheses. The floats are separated by commas or spaces. In 1D, one floating point number without parentheses is sufficient. Valid vectors are (1, 0, 2), (1e-4, -1e-3), and 1.
String	Strings are delimited by quotation marks. They are compatible with the C language format for strings. The following strings are valid: "Vdd", "output/diode".
Identifier	These are used to name objects such as nodes, devices, or attributes. They are compatible with the C language format for identifiers. The following identifiers are valid: Vdd, diode, bjt_345.
Assignment	These are used to set values to keywords. Therefore, the following are valid assignments: Digits=4, Save="output/diode".

B: Syntax

Table 118 Entry types in Sentaurus Device

Entry type	Description
Signal	Signals are time dependent, piecewise, linear functions (not to be confused with UNIX signals) that are defined as inputs on the contacts of a device. They are specified as follows: (value0 at time0, value1 at time1, ...value_n at time_n). The following signal is valid: (0 at 0, 1 at 10.0e-9, 1 at 20.0e-9).
List	Lists are collections of keywords, assignments, and complex entries. They are delimited by "(...)" or "{...}". The following lists are valid: { Number=0 Voltage=0 Voltage=(0 at 0, 0 at 2e-8) } { Method=Super Digits=6 Numerically } (MinStep=1e-15 InitialStep=1e-10 Digits=3)
Structured entries	These are parameterized definitions or commands that can have the forms: <keyword> {<keywords>}, <keyword> (<keywords>), or <keyword> (<list>) {<list>}.

APPENDIX C File-naming Convention

The data exchange format TDR or DF-ISE defines the file-naming convention for TCAD tools. The relevant items for Sentaurus Device are described here.

Overview

All strings that represent file names containing a dot (.) within their base name are taken literally. Otherwise, Sentaurus Device extends the given strings with the appropriate extension.

Sentaurus Device expands the extensions for output files by its tool extension `_des`, for example, the extension of a saved file is `_des.sav`.

Compressed files are additionally extended by the extension `.z`, for example, compressed plot files have the extension `_des.dat.z`. [Table 119](#) summarizes the *extensions* used in Sentaurus Device.

Table 119 Sentaurus Device file-naming convention

File	I/O	Extension
Command	I	<code>_des.cmd</code> , <code>.cmd</code>
Log	O	<code>_des.log</code>
Parameter	I	<code>.par</code>
Geometry	I	<code>.tdr</code> , <code>.grd</code>
Doping	I	<code>.tdr</code> , <code>.dat</code>
Lifetime	I	<code>.tdr</code> , <code>.dat</code>
Save	O	<code>_des.sav</code>
Load	I	<code>.sav</code>
Device Plot (grid-based)	O	<code>_des.tdr</code> , <code>_des.dat</code>
Current Plot	O	<code>_des.plt</code>
AC Extraction	O	<code>_ac_des.plt</code>
Montecarlo	I/O	See the Sentaurus SPARTA documentation in the <i>Sentaurus Device Monte Carlo User Guide</i> .

C: File-naming Convention

Compatibility with Old File-naming Convention

During transient simulations, quasistationaries, and continuations, the plot and save files are numbered by a global index.

Compatibility with Old File-naming Convention

To be compatible with the old file-naming conventions, Sentaurus Device tries to find the files according to the current convention and, if the corresponding files are not found, it tries to read the files according to the old convention. [Table 120](#) summarizes the old convention.

Table 120 Old file-naming convention

File	I/O	Extension
Command	I	.in
Output or Log	O	.out
Parameter	I	.par
Geometry	I	.geo
Doping	I	.dop
Lifetime	I	.life
Save	O	.sav
Load	I	.sav
Device Plot (grid-based)	O	.prt
Current Plot	O	.cur
AC Extraction	O	.ac

This appendix lists the most useful command-line options available in Sentaurus Device.

Starting Sentaurus Device

To start Sentaurus Device, enter:

```
sdevice [<options>] [<commandfile>]
```

Sentaurus Device appends automatically the corresponding extension to the given command file if necessary. If no command file is specified, Sentaurus Device reads from standard input.

Command-Line Options

Sentaurus Device interprets the following options:

- d Prints debug information into the `debug` file. The information printed includes the numeric values of the Jacobian and RHS for each equation at each solution step.
- h Lists these options and exits.
- i Prints the initial solution in the `save` file, and print files specified in the `File` section of the command file, and exits without performing further computations.
- n Does not include Newton information in the `log` file.
- P Writes the silicon model parameters into a file `models.par` and exits. This file can be modified and reloaded into Sentaurus Device to make customized changes to physical models and parameters.
- P:<Material> Writes the model parameters for the given material into a file `models.par` and exits.

D: Command-Line Options

Command-Line Options

-P:<Material>:<x>	Writes the model parameters for the given material and mole fraction into a file models.par and exits.
-P:<Material>:<x>:<y>	Writes the model parameters for the given material and mole fractions into a file models.par and exits.
-P:All	Writes the model parameters for all materials into a file models.par and exits.
-P:<Material>/<Material>	Writes the model parameters for the given material interface into a file models.par and exits.
-P <commandfile>	Writes the model parameters for the materials and interfaces used in <commandfile> into a file models.par and exits.
-L	Writes the silicon model parameters into the file Silicon.par and exits.
-L:<Material>	Writes a model parameter file <Material>.par for the specified material and exits.
-L:<Material>:<x>	Writes the model parameters for the given material and mole fraction into a file <Material>.par and exits.
-L:<Material>:<x>:<y>	Writes the model parameters for the given material and mole fractions into a file <Material>.par and exits.
-L:All	Writes a separate model parameter file for all materials and exits.
-L:<Material>/<Material>	Writes a model parameter file <Material>%<Material>.par for the specified material interface and exits.
-L <commandfile>	Writes model parameter files for all the materials, material interfaces, and electrodes used in <commandfile> and exits.
-M <commandfile>	Writes a parameter file models-M.par for regions with computed mole fraction dependencies.
-r	When used with -L or -P, reads parameters from the material library to generate output.
-q	Quiet mode for output.

-S	Writes the SiC model parameters into a <code>models_SiC.par</code> file and exits. This file can be modified and reloaded into Sentaurus Device to make customized changes to physical models and parameters.
-v	Prints header with version number of Sentaurus Device.
--parameter-names	Prints the names of the parameters from the parameter file that can be ramped. If a command file is also supplied, Sentaurus Device prints the parameters from the command file that can be ramped.
--exit-on-failure	Terminates immediately after a failed solve command.
--field-names	Prints fields and their numeric indices for use in the PMI.
--compiler-version	Prints the version of the C++ compiler that was used to compile Sentaurus Device.
--tcl	Invokes the Tcl interpreter to evaluate the command file.

In addition, generic tool options can be found in the relevant section of the *Installation Guide*.

D: Command-Line Options

Command-Line Options

This appendix presents information about obtaining run-time statistics from Sentaurus Device.

The sdevicestat Command

The command `sdevicestat` displays some statistics of a previous run of Sentaurus Device based on the information found in its `log` file. For example, the command:

```
sdevicestat test_des.log
```

generates the following statistics:

Total number of Newton iterations	:	8
Number of restarts	:	0
Rhs-time	:	9.19 % (38.80 s)
Jacobian-time	:	1.43 % (6.05 s)
Solve-time	:	88.76 % (374.70 s)
Overhead	:	0.62 % (2.61 s)
Total CPU time (sum of above times)	:	422.16 s

`sdevicestat` recognizes whether the `WallClock` keyword has been specified in the `Math` section of the Sentaurus Device command file (see [Parallelization on page 102](#)). In this case, the same simulation on a dual processor machine may produce the following output:

Total number of Newton iterations	:	8
Number of restarts	:	0
Rhs-time	:	8.57 % (20.68 s)
Jacobian-time	:	1.96 % (4.73 s)
Solve-time	:	88.33 % (213.07 s)
Overhead	:	1.14 % (2.74 s)
Total wallclock time (sum of above times):	241.22 s	

E: Run-Time Statistics

The sdevicestat Command

This appendix provides information about data and plot names.

Overview

[Table 121 on page 1060](#), [Table 122 on page 1084](#), and [Table 123 on page 1085](#) list the plot names that are recognized in a `Plot` section of Sentaurus Device (see [Plot Section on page 58](#)) and the data names that are available in the current plot PMI (see [Current Plot File of Sentaurus Device on page 1027](#)). If the plot name is empty, the data name in quotation marks can be used, for example:

```
Plot {  
    "eTemperatureRelaxationTime"  
}
```

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  
}
```

Special vector data can be plotted by appending `/SpecialVector` to the corresponding keyword, for example:

```
Plot {  
    TailDistribution/SpecialVector  
}
```

NOTE The location *rivertex* refers to region-interface vertices.

F: Data and Plot Names

Scalar Data

Scalar Data

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
AccepMinusConcentration		vertex	Material and Doping Specification on page 123	cm^{-3}
AcceptorConcentration	AcceptorConcentration	vertex		cm^{-3}
AlphaChargeDensity	AlphaCharge	vertex	Alpha Particles on page 490	cm^{-3}
AlphaGeneration		vertex	G^{Alpha} , Eq. 475, p. 491	$\text{cm}^{-3} \text{s}^{-1}$
AntimonyActiveConcentration		vertex	Material and Doping Specification on page 123	cm^{-3}
AntimonyConcentration	AntimonyConcentration	vertex	Sb, Material and Doping Specification on page 123	cm^{-3}
AntimonyPlusConcentration	sbPlus	vertex	Sb^+ , Chapter 6	cm^{-3}
ArsenicActiveConcentration		vertex	Material and Doping Specification on page 123	cm^{-3}
ArsenicConcentration	ArsenicConcentration	vertex	As, Material and Doping Specification on page 123	cm^{-3}
ArsenicPlusConcentration	AsPlus	vertex	As^+ , Chapter 6	cm^{-3}
AugerRecombination	AugerRecombination	vertex	R^A , Eq. 313, p. 322	$\text{cm}^{-3} \text{s}^{-1}$
AvalancheGeneration	AvalancheGeneration	vertex	G^{\parallel} , Eq. 316, p. 324	$\text{cm}^{-3} \text{s}^{-1}$
Band2BandGeneration	Band2Band	vertex	$G_{\text{net}}^{\text{bb}}$, Band-to-Band Tunneling Models on page 335	$\text{cm}^{-3} \text{s}^{-1}$

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
BandGap	BandGap	vertex	E_g , Bandgap and Electron-Affinity Models on page 226	eV
BandgapNarrowing	BandGapNarrowing	vertex	E_{bgn} , Bandgap and Electron-Affinity Models on page 226	eV
BeamGeneration	OptBeam	vertex	G^{opt} , Eq. 432, p. 436	$\text{cm}^{-3} \text{s}^{-1}$
BoronActiveConcentration		vertex	Material and Doping Specification on page 123	cm^{-3}
BoronConcentration	BoronConcentration	vertex	B, Material and Doping Specification on page 123	cm^{-3}
BoronMinusConcentration	bMinus	vertex	B ⁻ , Chapter 6	cm^{-3}
BuiltinPotential		vertex	ϕ_0 , Eq. 104, p. 206, Eq. 107, p. 207, Eq. 108, p. 208	V
CDL1Recombination	CDL1	vertex	R_1 , Coupled Defect Level (CDL) Recombination on page 320	$\text{cm}^{-3} \text{s}^{-1}$
CDL2Recombination	CDL2	vertex	R_2 , Coupled Defect Level (CDL) Recombination on page 320	$\text{cm}^{-3} \text{s}^{-1}$
CDLcRecombination	CDL3	vertex	$R - R_1 - R_2$, Coupled Defect Level (CDL) Recombination on page 320	$\text{cm}^{-3} \text{s}^{-1}$

F: Data and Plot Names

Scalar Data

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
CDLRecombination	CDL	vertex	R , Coupled Defect Level (CDL) Recombination on page 320	$\text{cm}^{-3} \text{s}^{-1}$
	ComplexRefractiveIndex	vertex	Complex Refractive Index Model on page 477	1
ConductionBandEnergy	ConductionBandEnergy	vertex	E_C , Eq. 82, p. 201	eV
ConductionCurrentDensity	ConductionCurrent	vertex	$ \vec{j}_n + \vec{j}_p $, Eq. 27, p. 178 or $ \vec{j}_M $ in metals, Eq. 67, p. 193	Acm^{-2}
CurECImACGreenFunction		vertex	Table 85 on page 507	1
CurECReACGreenFunction		vertex	Table 85 on page 507	1
CurETImACGreenFunction		vertex	Table 85 on page 507	V^{-1}
CurETReACGreenFunction		vertex	Table 85 on page 507	V^{-1}
CurHCImACGreenFunction		vertex	Table 85 on page 507	1
CurHCreACGreenFunction		vertex	Table 85 on page 507	1
CurHTImACGreenFunction		vertex	Table 85 on page 507	V^{-1}
CurHTReACGreenFunction		vertex	Table 85 on page 507	V^{-1}
CurLTImACGreenFunction		vertex	Table 85 on page 507	V^{-1}
CurLTReACGreenFunction		vertex	Table 85 on page 507	V^{-1}
CurPotImACGreenFunction		vertex	Table 85 on page 507	s^{-1}

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
CurPotReACGreenFunction		vertex	Table 85 on page 507	s ⁻¹
CurrentPotential	CurrentPotential	vertex	W , Current Potential on page 199	Acm ⁻¹
DeepLevels	DeepLevels	vertex	Energetic and Spatial Distribution of Traps on page 350	cm ⁻³
DielectricConstant		element	ϵ , Eq. 26, p. 178	1
DielectricConstant		vertex	ϵ , Eq. 26, p. 178	1
DielectricConstantAniso		element	ϵ_{aniso} , Anisotropic Electrical Permittivity on page 584	1
DielectricConstantAniso		vertex	ϵ_{aniso} , Anisotropic Electrical Permittivity on page 584	1
DisplacementCurrentDensity	DisplacementCurrent	vertex	$ \vec{j}_D $	Acm ⁻²
DonorConcentration	DonorConcentration	vertex	Material and Doping Specification on page 123	cm ⁻³
DonorPlusConcentration		vertex		cm ⁻³
DopingConcentration	Doping	vertex		cm ⁻³
eAlphaAvalanche		vertex	α_n , Eq. 316, p. 324	cm ⁻¹
eAmorphousRecombination	eGapStatesRecombination	vertex	Chapter 10	cm ⁻³ s ⁻¹
eAmorphousTrappedCharge	eTrappedCharge	vertex	Chapter 10	cm ⁻³
eAugerRecombination		vertex	R_n^A , Eq. 313, p. 322	cm ⁻³ s ⁻¹
eAvalancheGeneration	eAvalanche	vertex	G_n , Eq. 316, p. 324	cm ⁻³ s ⁻¹

F: Data and Plot Names

Scalar Data

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
eBand2BandGeneration	eBand2BandGeneration	vertex	Dynamic Nonlocal Path Band-to-Band Model on page 339	$\text{cm}^{-3} \text{s}^{-1}$
eCDL1Lifetime	eCDL1lifetime	vertex	τ_{n1} , Coupled Defect Level (CDL) Recombination on page 320	s
eCDL2Lifetime	eCDL2lifetime	vertex	τ_{n2} , Coupled Defect Level (CDL) Recombination on page 320	s
eCurrentDensity	eCurrent	vertex	$ \vec{J}_n $, Eq. 27, p. 178	Acm^{-2}
eDensity	eDensity	vertex	n , Eq. 26, p. 178	cm^{-3}
eDifferentialGain	eDifferentialGain	vertex	Stimulated and Spontaneous Emission Coefficients on page 809	cm^2
eDirectTunnelCurrent	eDirectTunneling	vertex	Direct Tunneling on page 514	Acm^{-2}
eDriftVelocity	eDriftVelocity	vertex	Electron drift velocity	cm s^{-1}
eeDiffusionLNS		vertex	Table 85 on page 507	$\text{C}^2 \text{s}^{-1} \text{cm}^{-1}$
eEffectiveField	eEffectiveField	vertex	E_n^{eff} , Eq. 331, p. 331	Vcm^{-1}
eEffectiveStateDensity		vertex	N_C , Effective Masses and Effective Density-of-States on page 237	cm^{-3}
eeFlickerGRLNS		vertex	Table 85 on page 507	$\text{C}^2 \text{s}^{-1} \text{cm}^{-1}$
eeMonopolarGRLNS		vertex	Table 85 on page 507	$\text{C}^2 \text{s}^{-1} \text{cm}^{-1}$

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
eEnormal	eEnormal	vertex	F_{\perp} , Eq. 240, p. 286 or $F_{n,\perp}$, Eq. 241, p. 287	Vcm ⁻¹
eEparallel	eEparallel	vertex	F_n , Eq. 270, p. 300	Vcm ⁻¹
eEquilibriumDensity	eEquilibriumDensity	vertex	n , Eq. 26, p. 178 at zero applied voltages (zero currents)	cm ⁻³
EffectiveBandGap	EffectiveBandGap	vertex	$E_g - E_{bgn}$, Bandgap and Electron-Affinity Models on page 226	eV
EffectiveIntrinsicDensity	EffectiveIntrinsicDensity	vertex	$n_{i,eff}$, Eq. 129, p. 225	cm ⁻³
eFreeCarrierAbsorption	eFreeCarrierAbsorption	vertex	$\alpha_n n \Psi ^2$, Eq. 741, p. 702	cm ⁻³
eGradQuasiFermi	eGradQuasiFermi	vertex	$ \nabla \Phi_n $, Eq. 271, p. 300	Vcm ⁻¹
eHeatFlux	eHeatFlux	vertex	$ \vec{S}_n $, Eq. 41, p. 186	Wcm ⁻²
eInterfaceTrappedCharge		vertex	Chapter 10	cm ⁻²
eIonIntegral	eIonIntegral	vertex	Approximate Breakdown Analysis: Poisson Equation Approach on page 333	1
eJouleHeat	eJouleHeat	vertex	Table 21 on page 181	Wcm ⁻³
ElectricField	ElectricField	vertex	F	Vcm ⁻¹
ElectronAffinity	ElectronAffinity	vertex	χ , Bandgap and Electron-Affinity Models on page 226	eV
ElectrostaticPotential	Potential	vertex	ϕ , Eq. 26, p. 178	V

F: Data and Plot Names

Scalar Data

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
eLifetime	eLifeTime	vertex	τ_n , Eq. 278, p. 310	s
EMLABGeneration		vertex	G^{EMW} , Finite-Difference Time-Domain Method on page 447	$\text{cm}^{-3} \text{s}^{-1}$
eMobility	eMobility	element	μ_n , Chapter 8	$\text{cm}^2 \text{V}^{-1} \text{s}^{-1}$
eMobility	eMobility	vertex	μ_n , Chapter 8	$\text{cm}^2 \text{V}^{-1} \text{s}^{-1}$
eMobilityAniso		element	μ_n^{aniso} , Anisotropic Mobility on page 576	$\text{cm}^2 \text{V}^{-1} \text{s}^{-1}$
eMobilityAniso		vertex	μ_n^{aniso} , Anisotropic Mobility on page 576	$\text{cm}^2 \text{V}^{-1} \text{s}^{-1}$
eMobilityAnisoFactor		vertex	r_e , Eq. 593, p. 576	1
eMobilityStressFactorXX	eMobilityStressFactorXX	vertex	Using Piezoresistance Mobility Model on page 608	1
eMobilityStressFactorXY	eMobilityStressFactorXY	vertex		1
eMobilityStressFactorXZ	eMobilityStressFactorXZ	vertex		1
eMobilityStressFactorYY	eMobilityStressFactorYY	vertex		1
eMobilityStressFactorYZ	eMobilityStressFactorYZ	vertex		1
eMobilityStressFactorZZ	eMobilityStressFactorZZ	vertex		1
eNLITunnelingGeneration	eBarrierTunneling	vertex	Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 518	$\text{cm}^{-3} \text{s}^{-1}$
eQuantumPotential	eQuantumPotential	vertex	Λ_n , Eq. 174, p. 251	eV
eQuasiFermiPotential	eQuasiFermi	vertex	Φ_n , Quasi-Fermi Potential on page 201	V

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
EquilibriumPotential	EquilibriumPotential	vertex	ϕ , Eq. 26, p. 178 at zero applied voltages (zero currents)	V
eRelativeEffectiveMass		vertex	m_n , Effective Masses and Effective Density-of-States on page 237	1
eSaturationVelocity		vertex	$v_{sat,n}$, Velocity Saturation Models on page 299	cm s^{-1}
eSaturationVelocityAniso		vertex	$v_{sat,n}^{\text{aniso}}$, Anisotropic Mobility on page 576	cm s^{-1}
eSchenkBGN	eSchenkBGN	vertex	$-\Lambda_n$, Eq. 174, p. 251	eV
eTemperature	eTemperature	vertex	T_n , Hydrodynamic Transport Model on page 184	K
eTemperatureRelaxationTime		vertex	τ_{en} , Eq. 51, p. 187	s
eTensorMobilityFactorXX	eTensorMobilityFactorXX	vertex	Chapter 23	1
eTensorMobilityFactorYY	eTensorMobilityFactorYY	vertex	Chapter 23	1
eTensorMobilityFactorZZ	eTensorMobilityFactorZZ	vertex	Chapter 23	1
eTensorMobilityXX	eTensorMobilityXX	vertex	Chapter 23	$\text{cm}^2/(\text{Vs})$
eTensorMobilityYY	eTensorMobilityYY	vertex	Chapter 23	$\text{cm}^2/(\text{Vs})$
eTensorMobilityZZ	eTensorMobilityZZ	vertex	Chapter 23	$\text{cm}^2/(\text{Vs})$
eThermoElectricPower	eThermoelectricPower	vertex	P_n , Eq. 708, p. 649	V K^{-1}
eVelocity	eVelocity	vertex	$v_n = \vec{J}_n/nq $	cm s^{-1}

F: Data and Plot Names

Scalar Data

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
f1BandOccupancy001	f1BandOccupancy001	vertex	Using Intel Mobility Model on page 628	1
f1BandOccupancy010	f1BandOccupancy010	vertex		1
f1BandOccupancy100	f1BandOccupancy100	vertex		1
f2BandOccupancy001	f2BandOccupancy001	vertex		1
f2BandOccupancy010	f2BandOccupancy010	vertex		1
f2BandOccupancy100	f2BandOccupancy100	vertex		1
FowlerNordheim	FowlerNordheim	vertex	j_{FN} , Eq. 502, p. 513	Acm^{-2}
Grad2PoECACGreenFunction		vertex	Table 85 on page 507	$\text{V}^2 \text{s}^2 \text{C}^{-2} \text{cm}^{-2}$
Grad2PoHCACGreenFunction		vertex	Table 85 on page 507	$\text{V}^2 \text{s}^2 \text{C}^{-2} \text{cm}^{-2}$
hAlphaAvalanche		vertex	α_p , Eq. 316, p. 324	cm^{-1}
hAmorphousRecombination	hGapStatesRecombination	vertex	Chapter 10	$\text{cm}^{-3} \text{s}^{-1}$
hAmorphousTrappedCharge	hTrappedCharge	vertex	Chapter 10	cm^{-3}
hAugerRecombination		vertex	R_p^A , Eq. 313, p. 322	$\text{cm}^{-3} \text{s}^{-1}$
hAvalancheGeneration	hAvalanche	vertex	G_p , Eq. 316, p. 324	$\text{cm}^{-3} \text{s}^{-1}$
hBand2BandGeneration	hBand2BandGeneration	vertex	Dynamic Nonlocal Path Band-to-Band Model on page 339	$\text{cm}^{-3} \text{s}^{-1}$
hCDL1Lifetime	hCDL1lifetime	vertex	τ_{p1} , Coupled Defect Level (CDL) Recombination on page 320	s
hCDL2Lifetime	hCDL2lifetime	vertex	τ_{p2} , Coupled Defect Level (CDL) Recombination on page 320	s
hCurrentDensity	hCurrent	vertex	$ j_p $, Eq. 27, p. 178	Acm^{-2}

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
hDensity	hDensity	vertex	p , Eq. 26, p. 178	cm^{-3}
hDifferentialGain	hDifferentialGain	vertex	Stimulated and Spontaneous Emission Coefficients on page 809	cm^2
hDirectTunnelCurrent	hDirectTunneling	vertex	Direct Tunneling on page 514	Acm^{-2}
hDriftVelocity	hDriftVelocity	vertex	Hole drift velocity	cm s^{-1}
HeavyIonChargeDensity	HeavyIonChargeDensity	vertex	Heavy Ions on page 492	cm^{-3}
HeavyIonGeneration		vertex	G^{HeavyIon} , Eq. 480, p. 493	$\text{cm}^{-3} \text{s}^{-1}$
hEffectiveField	hEffectiveField	vertex	E_p^{eff} , Eq. 332, p. 331	Vcm^{-1}
hEffectiveStateDensity		vertex	N_V , Effective Masses and Effective Density-of-States on page 237	cm^{-3}
heiTemperature	HEItemperature	vertex	T_{hei} , hot-electron temperature, computed as postprocessing approach (Carrier TempPost), Chapter 18	K
hEnormal	hEnormal	vertex	F_{\perp} , Eq. 240, p. 286 or $F_{p_2\perp}$, Eq. 241, p. 287	Vcm^{-1}
hEparallel	hEparallel	vertex	F_p , Eq. 270, p. 300	Vcm^{-1}
hEquilibriumDensity	hEquilibriumDensity	vertex	p , Eq. 26, p. 178 at zero applied voltages (zero currents)	cm^{-3}
hFreeCarrierAbsorption	hFreeCarrierAbsorption	vertex	$\alpha_p p \Psi ^2$, Eq. 741, p. 702	cm^{-3}

F: Data and Plot Names

Scalar Data

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
hGradQuasiFermi	hGradQuasiFermi	vertex	$ \nabla\Phi_p $, Eq. 271, p. 300	Vcm ⁻¹
hhDiffusionLNS		vertex	Table 85 on page 507	C ² s ⁻¹ cm ⁻¹
hHeatFlux	hHeatFlux	vertex	$ \vec{S}_p $, Eq. 42, p. 186	Wcm ⁻²
hhFlickerGRLNS		vertex	Table 85 on page 507	C ² s ⁻¹ cm ⁻¹
hhMonopolarGRLNS		vertex	Table 85 on page 507	C ² s ⁻¹ cm ⁻¹
hInterfaceTrappedCharge		vertex	Chapter 10	cm ⁻²
hIonIntegral	hIonIntegral	vertex	Approximate Breakdown Analysis: Poisson Equation Approach on page 333	1
hJouleHeat	hJouleHeat	vertex	Table 21 on page 181	Wcm ⁻³
hLifetime	hLifeTime	vertex	τ_p , Eq. 278, p. 310	s
hMobility	hMobility	element	μ_p , Chapter 8	cm ² V ⁻¹ s ⁻¹
hMobility	hMobility	vertex	μ_p , Chapter 8	cm ² V ⁻¹ s ⁻¹
hMobilityAniso		element	μ_p^{aniso} , Anisotropic Mobility on page 576	cm ² V ⁻¹ s ⁻¹
hMobilityAniso		vertex	μ_p^{aniso} , Anisotropic Mobility on page 576	cm ² V ⁻¹ s ⁻¹
hMobilityAnisoFactor		vertex	r_h , Eq. 593, p. 576	1

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
hMobilityStressFactorXX	hMobilityStressFactorXX	vertex	Using Piezoresistance Mobility Model on page 608	1
hMobilityStressFactorXY	hMobilityStressFactorXY	vertex		1
hMobilityStressFactorXZ	hMobilityStressFactorXZ	vertex		1
hMobilityStressFactorYY	hMobilityStressFactorYY	vertex		1
hMobilityStressFactorYZ	hMobilityStressFactorYZ	vertex		1
hMobilityStressFactorZZ	hMobilityStressFactorZZ	vertex		1
hNLITunnelingGeneration	hBarrierTunneling	vertex	Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 518	$\text{cm}^{-3} \text{s}^{-1}$
HotElectronInj	HotElectronInjection	vertex, rivertex	Hot-electron current density j_{he} at interface, Eq. 534, p. 535, Eq. 540, p. 537	Acm^{-2}
HotHoleInj	HotHoleInjection	vertex, rivertex	Hot-hole current density j_{hh} at interface, Eq. 534, p. 535, Eq. 540, p. 537	Acm^{-2}
hQuantumPotential	hQuantumPotential	vertex	Λ_p , Eq. 174, p. 251	eV
hQuasiFermiPotential	hQuasiFermi	vertex	Φ_p , Quasi-Fermi Potential on page 201	V
hRelativeEffectiveMass		vertex	m_p , Effective Masses and Effective Density-of-States on page 237	1
hSaturationVelocity		vertex	$v_{sat,p}$, Velocity Saturation Models on page 299	cm s^{-1}
hSaturationVelocityAniso		vertex	$v_{sat,p}^{\text{aniso}}$, Anisotropic Mobility on page 576	cm s^{-1}

F: Data and Plot Names

Scalar Data

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
hSchenkBGN	hSchenkBGN	vertex	$-\Lambda_p$, Eq. 174, p. 251	eV
hTemperature	hTemperature	vertex	T_p , Hydrodynamic Transport Model on page 184	K
hTemperatureRelaxationTime		vertex	τ_{ep} , Eq. 52, p. 187	s
hTensorMobilityFactorXX	hTensorMobilityFactorXX	vertex	Chapter 23	1
hTensorMobilityFactorYY	hTensorMobilityFactorYY	vertex	Chapter 23	1
hTensorMobilityFactorZZ	hTensorMobilityFactorZZ	vertex	Chapter 23	1
hTensorMobilityXX	hTensorMobilityXX	vertex	Chapter 23	$\text{cm}^2/(\text{Vs})$
hTensorMobilityYY	hTensorMobilityYY	vertex	Chapter 23	$\text{cm}^2/(\text{Vs})$
hTensorMobilityZZ	hTensorMobilityZZ	vertex	Chapter 23	$\text{cm}^2/(\text{Vs})$
hThermoElectricPower	hThermoelectricPower	vertex	P_p , Eq. 709, p. 649	V K^{-1}
hVelocity	hVelocity	vertex	$v_p = \vec{J}_p/pq $	cm s^{-1}
HydrogenAtom	HydrogenAtom	vertex	Hydrogen Transport Degradation Model on page 381	cm^{-3}
HydrogenIon	HydrogenIon	vertex	Hydrogen Transport Degradation Model on page 381	cm^{-3}
HydrogenMolecule	HydrogenMolecule	vertex	Hydrogen Transport Degradation Model on page 381	cm^{-3}
ImConductionCurrentResponse		vertex	$\text{Im}\left(\frac{z}{J_n + J_p}\right)$ AC Current Density Responses on page 895	$\text{A cm}^{-2}\text{V}^{-1}$

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
ImDisplacementCurrentResponse		vertex	$\text{Im}(\tilde{z})$ AC Current Density Responses on page 895	$\text{Acm}^{-2}\text{V}^{-1}$
ImeCurrentResponse		vertex	$\text{Im}(\tilde{J}_n)$ AC Current Density Responses on page 895	$\text{Acm}^{-2}\text{V}^{-1}$
ImeDensityResponse		vertex	$\text{Im}(\tilde{n})$ AC Response on page 893	$\text{cm}^{-3}\text{V}^{-1}$
ImeeDiffusionLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
ImeeFlickerGRLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
ImeeLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
ImeeMonopolarGRLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
ImeEnFluxResponse		vertex	$\text{Im}(\tilde{S}_n)$ AC Current Density Responses on page 895	$\text{Wcm}^{-2}\text{V}^{-1}$
ImElectrostaticPotentialResponse		vertex	$\text{Im}(\tilde{\phi})$ AC Response on page 893	1
ImeTemperatureResponse		vertex	$\text{Im}(\tilde{T}_n)$ AC Response on page 893	KV^{-1}

F: Data and Plot Names

Scalar Data

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
ImhCurrentResponse		vertex	$\text{Im}(\vec{J}_p)$ AC Current Density Responses on page 895	$\text{Acm}^{-2}\text{V}^{-1}$
ImhDensityResponse		vertex	$\text{Im}(\tilde{p})$ AC Response on page 893	$\text{Acm}^{-2}\text{V}^{-1}$
ImhEnFluxResponse		vertex	$\text{Im}(\vec{S}_p)$ AC Current Density Responses on page 895	$\text{Wcm}^{-2}\text{V}^{-1}$
ImhhDiffusionLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
ImhhFlickerGRLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
ImhhLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
ImhhMonopolarGRLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
ImhTemperatureResponse		vertex	$\text{Im}(\tilde{T}_p)$ AC Response on page 893	KV^{-1}
ImLatticeTemperatureResponse		vertex	$\text{Im}(\tilde{T})$ AC Response on page 893	KV^{-1}
ImlEnFluxResponse		vertex	$\text{Im}(\vec{S}_L)$ AC Current Density Responses on page 895	$\text{Wcm}^{-2}\text{V}^{-1}$
ImLNISD		vertex	Table 85 on page 507	$\text{A}^2\text{scm}^{-3}$
ImLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
ImTotalCurrentResponse		vertex	$\text{Im}\left(\frac{\dot{z}}{J_n} + \frac{\dot{z}}{J_p} + \frac{\dot{z}}{J_D}\right)$ AC Current Density Responses on page 895	$\text{Acm}^{-2}\text{V}^{-1}$
ImTrapLNISD		vertex	Table 85 on page 507	$\text{A}^2\text{scm}^{-3}$
ImTrapLNVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
IndiumActiveConcentration		vertex	Material and Doping Specification on page 123	cm^{-3}
IndiumConcentration	IndiumConcentration	vertex	In, Material and Doping Specification on page 123	cm^{-3}
IndiumMinusConcentration	inMinus	vertex	In ⁻ , Chapter 6	cm^{-3}
IntrinsicDensity	IntrinsicDensity	vertex	n_i , Eq. 128, p. 225	cm^{-3}
LaserIntensity	LaserIntensity	vertex	Combined laser intensity for multimode lasing	Wcm^{-3}
LatticeHeatCapacity		vertex	c_L , Heat Capacity on page 647	$\text{JK}^{-1}\text{cm}^{-3}$
LatticeTemperature	LatticeTemperature, Temperature	vertex	T, Uniform Self-Heating on page 182	K
lHeatFlux	lHeatFlux	vertex	$ \vec{S}_L $, Eq. 43, p. 186	Wcm^{-2}
MeanIonIntegral	MeanIonIntegral	vertex	Approximate Breakdown Analysis: Poisson Equation Approach on page 333	1

F: Data and Plot Names

Scalar Data

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
MatGain	MatGain	vertex	Laser material gain, Stimulated and Spontaneous Emission Coefficients on page 809	m^{-1}
MobilityAcceptorConcentration		vertex	Mobility Doping File on page 304	cm^{-3}
MobilityDonorConcentration		vertex	Mobility Doping File on page 304	cm^{-3}
NDopantActiveConcentration		vertex	Material and Doping Specification on page 123	cm^{-3}
NDopantConcentration	NdopantConcentration	vertex	NDopant, Material and Doping Specification on page 123	cm^{-3}
NDopantPlusConcentration	NdopantPlus	vertex	NDopant ⁺ , Chapter 6	cm^{-3}
NitrogenActiveConcentration		vertex	Material and Doping Specification on page 123	cm^{-3}
NitrogenConcentration	NitrogenConcentration	vertex	N, Material and Doping Specification on page 123	cm^{-3}
NitrogenPlusConcentration	NitrogenPlus	vertex	N ⁺ , Chapter 6	cm^{-3}
OneOverDegradationTime		vertex	Chapter 12	s^{-1}
OpticalField	OpticalField	vertex	Plot optical intensity as well as real and imaginary parts of optical field. Requires TDR grid file.	W/m^3 V/m
OpticalGeneration	OpticalGeneration	vertex	G_0^{opt} , Eq. 438, p. 440	$cm^{-3}s^{-1}$

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
OpticalIntensityMode0... OpticalIntensityMode9	OpticalIntensityMode0 ... OpticalIntensityMode9	vertex	Current File and Plot Variables for Laser Simulation on page 681	Wcm ⁻³
OpticalPolarizationAngleMode0 ... OpticalPolarizationAngleMode9	OpticalPolarizationAngleMode0... OpticalPolarizationAngleMode9	vertex	Polarization-dependent Optical Matrix Element on page 822	1
PDopantActiveConcentration		vertex	Material and Doping Specification on page 123	cm ⁻³
PDopantConcentration	pDopantConcentration	vertex	PDopant, Material and Doping Specification on page 123	cm ⁻³
PDopantMinusConcentration	pDopantMinus	vertex	PDopant-, Chapter 6	cm ⁻³
PE_Charge	PE_Charge	vertex	q_{PE} , Eq. 985, p. 1013	cm ⁻³
PeltierHeat	PeltierHeat	vertex	Table 21 on page 181	Wcm ⁻³
PhosphorusActiveConcentration		vertex	Material and Doping Specification on page 123	cm ⁻³
PhosphorusConcentration	PhosphorusConcentration	vertex	P, Material and Doping Specification on page 123	cm ⁻³
PhosphorusPlusConcentration	phPlus	vertex	P ⁺ , Chapter 6	cm ⁻³
PMIRecombination	PMIRecombination	vertex	R^{PMI} , Generation-Recombination Model on page 933	cm ⁻³ s ⁻¹

F: Data and Plot Names

Scalar Data

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
PMIUserField0	PMIUserField0	vertex	Command File of Sentaurus Device on page 916	1
PMIUserField1	PMIUserField1	vertex		1
PMIUserField2	PMIUserField2	vertex		1
PMIUserField3	PMIUserField3	vertex		1
PMIUserField4	PMIUserField4	vertex		1
PMIUserField5	PMIUserField5	vertex		1
PMIUserField6	PMIUserField6	vertex		1
PMIUserField7	PMIUserField7	vertex		1
PMIUserField8	PMIUserField8	vertex		1
PMIUserField9	PMIUserField9	vertex		1
PoECImACGreenFunction		vertex	Table 85 on page 507	VsC ⁻¹
PoECReACGreenFunction		vertex	Table 85 on page 507	VsC ⁻¹
PoETImACGreenFunction		vertex	Table 85 on page 507	A ⁻¹
PoETReACGreenFunction		vertex	Table 85 on page 507	A ⁻¹
PoHCImACGreenFunction		vertex	Table 85 on page 507	VsC ⁻¹
PoHCReACGreenFunction		vertex	Table 85 on page 507	VsC ⁻¹
PoHTImACGreenFunction		vertex	Table 85 on page 507	A ⁻¹
PoHTReACGreenFunction		vertex	Table 85 on page 507	A ⁻¹
PoLTImACGreenFunction		vertex	Table 85 on page 507	A ⁻¹
PoLTReACGreenFunction		vertex	Table 85 on page 507	A ⁻¹
PoPotImACGreenFunction		vertex	Table 85 on page 507	VC ⁻¹

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
PoPotReACGreenFunction		vertex	Table 85 on page 507	VC^{-1}
Polarization	Polarization	vertex	$ \vec{P} $, Chapter 22	Ccm^{-2}
QW_eLeakageCurrent	QW_eLeakageCurrent	vertex	Thermionic Emission on page 805	Acm^{-2}
QW_eNetCapture	QW_eNetCapture	vertex	C for electrons, Eq. 816, p. 807	$\text{cm}^{-3} \text{s}^{-1}$
QW_hLeakageCurrent	QW_hLeakageCurrent	vertex	Thermionic Emission on page 805	Acm^{-2}
QW_hNetCapture	QW_hNetCapture	vertex	C for holes, Eq. 816, p. 807	$\text{cm}^{-3} \text{s}^{-1}$
QWeDensity	QWeDensity	vertex	n , Eq. 854, p. 818	cm^{-3}
QWeQuasiFermi	QWeQuasiFermi	vertex	Φ_n , Quasi-Fermi Potential on page 201	V
QWhDensity	QWhDensity	vertex	p , Eq. 855, p. 818	cm^{-3}
QWhQuasiFermi	QWhQuasiFermi	vertex	Φ_p , Quasi-Fermi Potential on page 201	V
QuasiFermiPotential		vertex	Φ , Eq. 118, p. 213	V
RadiationGeneration		vertex	G_r , Eq. 430, p. 435	$\text{cm}^{-3} \text{s}^{-1}$
RadiativeRecombination	RadiativeRecombination	vertex	R , Eq. 312, p. 322	$\text{cm}^{-3} \text{s}^{-1}$
RecombinationHeat	RecombinationHeat	vertex	Table 21 on page 181	Wcm^{-3}
ReConductionCurrentResponse		vertex	$\text{Re}\left(\frac{\dot{z}}{J_n + J_p}\right)$ AC Current Density Responses on page 895	$\text{Acm}^{-2} \text{V}^{-1}$

F: Data and Plot Names

Scalar Data

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
ReDisplacementCurrentResponse		vertex	$\text{Re}(\tilde{\mathbf{z}}_{J_D})$ AC Current Density Responses on page 895	$\text{Acm}^{-2}\text{V}^{-1}$
ReeCurrentResponse		vertex	$\text{Re}(\tilde{\mathbf{z}}_{J_n})$ AC Current Density Responses on page 895	$\text{Acm}^{-2}\text{V}^{-1}$
ReeDensityResponse		vertex	$\text{Re}(\tilde{n})$ AC Response on page 893	$\text{Acm}^{-2}\text{V}^{-1}$
ReeeDiffusionLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
ReeeFlickerGRLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
ReeeLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
ReeeMonopolarGRLNVXVSD		vertex	Table 85 on page 507	$\text{V}^2\text{scm}^{-3}$
ReeEnFluxResponse		vertex	$\text{Re}(\tilde{\mathbf{z}}_{S_n})$ AC Current Density Responses on page 895	$\text{Wcm}^{-2}\text{V}^{-1}$
ReElectrostaticPotentialResponse		vertex	$\text{Re}(\tilde{\phi})$ AC Response on page 893	1
ReeTemperatureResponse		vertex	$\text{Re}(\tilde{T}_n)$ AC Response on page 893	KV^{-1}
RefractiveIndex		element	n , Refractive Index Models on page 472	1

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
RefractiveIndex		vertex	<i>n</i> , Refractive Index Models on page 472	1
RehCurrentResponse		vertex	$\text{Re}(\vec{J}_p)$ AC Current Density Responses on page 895	$\text{Acm}^{-2}\text{V}^{-1}$
RehDensityResponse		vertex	$\text{Re}(\tilde{p})$ AC Response on page 893	$\text{Acm}^{-2}\text{V}^{-1}$
RehEnFluxResponse		vertex	$\text{Re}(\vec{S}_n)$ AC Current Density Responses on page 895	$\text{Wcm}^{-2}\text{V}^{-1}$
RehhDiffusionLNVXVSD		vertex	Table 85 on page 507	V^2cm^{-3}
RehhFlickerGRLNVXVSD		vertex	Table 85 on page 507	V^2cm^{-3}
RehhLNVXVSD		vertex	Table 85 on page 507	V^2cm^{-3}
RehhMonopolarGRLNVXVSD		vertex	Table 85 on page 507	V^2cm^{-3}
RehTemperatureResponse		vertex	$\text{Re}(\tilde{T}_p)$ AC Response on page 893	KV^{-1}
ReLatticeTemperatureResponse		vertex	$\text{Re}(\tilde{T})$ AC Response on page 893	KV^{-1}
RelEnFluxResponse		vertex	$\text{Re}(\vec{S}_L)$ AC Current Density Responses on page 895	$\text{Wcm}^{-2}\text{V}^{-1}$

F: Data and Plot Names

Scalar Data

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
ReLNISD		vertex	Table 85 on page 507	A ² scm ⁻³
ReLNVXVSD		vertex	Table 85 on page 507	V ² scm ⁻³
ReTotalCurrentResponse		vertex	Re($\frac{z}{J_n} + \frac{z}{J_p} + \frac{z}{J_D}$) AC Current Density Responses on page 895	Acm ⁻² V ⁻¹
ReTrapLNISD		vertex	Table 85 on page 507	A ² scm ⁻³
ReTrapLNVSD		vertex	Table 85 on page 507	V ² scm ⁻³
SpaceCharge	SpaceCharge	vertex	Eq. 26, p. 178	cm ⁻³
SpontaneousRecombination	SpontaneousRecombination	vertex	Spontaneous Recombination Rate on page 812	cm ⁻³ s ⁻¹
SRHRecombination	SRHRecombination	vertex	$R_{\text{net}}^{\text{SRH}}$, Shockley–Read–Hall Recombination on page 309	cm ⁻³ s ⁻¹
StimulatedRecombination	StimulatedRecombination	vertex	Stimulated Recombination Rate on page 811	cm ⁻³ s ⁻¹
StressXX	Stressxx	vertex	Chapter 23	Pa
StressXY	Stressxy	vertex	Chapter 23	Pa
StressXZ	Stressxz	vertex	Chapter 23	Pa
StressYY	Stressyy	vertex	Chapter 23	Pa
StressYZ	Stressyz	vertex	Chapter 23	Pa
StressZZ	Stresszz	vertex	Chapter 23	Pa
SurfaceRecombination	SurfaceRecombination	vertex, rivertex	Surface SRH Recombination on page 319	cm ⁻² s ⁻¹
ThermalConductivity	ThermalConductivity	vertex	κ , Eq. 705, p. 648	Wcm ⁻¹ K ⁻¹

Table 121 Scalar data

Data name	Plot name	Location	Description	Unit
ThermalConductivityAniso		vertex	κ_{aniso} , Anisotropic Thermal Conductivity on page 586	$\text{Wcm}^{-1}\text{K}^{-1}$
ThomsonHeat	ThomsonHeat	vertex	Table 21 on page 181	Wcm^{-3}
TotalConcentration		vertex	Material and Doping Specification on page 123	cm^{-3}
TotalCurrentDensity	Current	vertex	$ \vec{J}_n + \vec{J}_p + \vec{J}_D $	Acm^{-2}
TotalHeat	TotalHeat	vertex	Sum of all heat generation terms, Uniform Self-Heating on page 182 , Hydrodynamic Transport Model on page 184 , Conductivity of Metals on page 193	Wcm^{-3}
TotalInterfaceTrapConcentration		vertex	Chapter 10	cm^{-2}
TotalRecombination	TotalRecombination	vertex	Sum of all generation–recombination terms, Chapter 9	$\text{cm}^{-3}\text{s}^{-1}$
TotalTrapConcentration		vertex	Chapter 10	cm^{-3}
UserSpeciesConcentration	UserSpeciesConcentration	vertex	User-Defined Species on page 125	cm^{-3}
UserSpeciesIncompleteConcentration	UserSpeciesIncompleteConcentration	vertex		cm^{-3}
ValenceBandEnergy	ValenceBandEnergy	vertex	E_V , Eq. 83, p. 201	eV
xMoleFraction	xMoleFraction	vertex	Mole-Fraction Specification on page 550	1
yMoleFraction	yMoleFraction	vertex		1

F: Data and Plot Names

Vector Data

Vector Data

Table 122 Vector data

Data name	Plot name	Location	Description	Unit
ConductionCurrentDensity	ConductionCurrent	vertex	$\vec{J}_n + \vec{J}_p$, Eq. 27, p. 178 or \vec{J}_M in metals, Eq. 67, p. 193	Acm^{-2}
DisplacementCurrentDensity	DisplacementCurrent	vertex	\vec{J}_D	Acm^{-2}
eCurrentDensity	eCurrent	vertex	\vec{J}_n , Eq. 27, p. 178	Acm^{-2}
eDriftVelocity	eDriftVelocity	vertex	Electron drift velocity	cm s^{-1}
eGradQuasiFermi	eGradQuasiFermi	vertex	$\nabla\Phi_n$, Eq. 271, p. 300	Vcm^{-1}
eHeatFlux	eHeatFlux	vertex	\vec{S}_n , Eq. 41, p. 186	Wcm^{-2}
ElectricField	ElectricField	vertex	\vec{F}	Vcm^{-1}
EquilibriumElectricField		vertex	\vec{F}_{eq} , Eq. 111, p. 209	Vcm^{-1}
eVelocity	eVelocity	vertex	$v_n = -\vec{J}_n/qn$	cm s^{-1}
GradPoECImACGreenFunction		vertex	Table 85 on page 507	$\text{VsC}^{-1}\text{cm}^{-1}$
GradPoECReACGreenFunction		vertex	Table 85 on page 507	$\text{VsC}^{-1}\text{cm}^{-1}$
GradPoETImACGreenFunction		vertex	Table 85 on page 507	$\text{A}^{-1}\text{cm}^{-1}$
GradPoETReACGreenFunction		vertex	Table 85 on page 507	$\text{A}^{-1}\text{cm}^{-1}$
GradPoHCImACGreenFunction		vertex	Table 85 on page 507	$\text{VsC}^{-1}\text{cm}^{-1}$
GradPoHCReACGreenFunction		vertex	Table 85 on page 507	$\text{VsC}^{-1}\text{cm}^{-1}$
GradPoHTImACGreenFunction		vertex	Table 85 on page 507	$\text{A}^{-1}\text{cm}^{-1}$
GradPoHTReACGreenFunction		vertex	Table 85 on page 507	$\text{A}^{-1}\text{cm}^{-1}$
hCurrentDensity	hCurrent	vertex	\vec{J}_p , Eq. 27, p. 178	Acm^{-2}
hDriftVelocity	hDriftVelocity	vertex	Hole drift velocity	cm s^{-1}
hGradQuasiFermi	hGradQuasiFermi	vertex	$\nabla\Phi_p$, Eq. 271, p. 300	Vcm^{-1}
hHeatFlux	hHeatFlux	vertex	\vec{S}_p , Eq. 42, p. 186	Wcm^{-2}
hVelocity	hVelocity	vertex	$v_p = \vec{J}_p qp$	cm s^{-1}
lHeatFlux	lHeatFlux	vertex	\vec{S}_L , Eq. 43, p. 186	Wcm^{-2}

Table 122 Vector data

Data name	Plot name	Location	Description	Unit
NonLocalBackDirection	NonLocal	vertex	Visualizing Nonlocal Meshes on page 105	μm
NonLocalDirection	NonLocal	vertex		μm
OpticalField	OpticalField	vertex	Plot optical intensity as well as real and imaginary parts of optical field. Requires TDR grid file.	W/m^{-3} V/m
PE_Polarization	PE_Polarization	vertex	P_{PE} , Eq. 985, p. 1013	Ccm^{-2}
Polarization	Polarization	element	\vec{P} , Chapter 22	Ccm^{-2}
Polarization	Polarization	vertex	\vec{P} , Chapter 22	Ccm^{-2}
TotalCurrentDensity	Current	vertex	$\vec{J}_n + \vec{J}_p + \vec{J}_D$	Acm^{-2}

Special Vector Data

Table 123 Special Vector data

Data name	Plot name	Location	Description	Unit
	TaileDistribution	vertex	Electron energy distribution. Tail Distribution Hot-Carrier Injection on page 537	1
	TailhDistribution	vertex	Hole energy distribution. Tail Distribution Hot-Carrier Injection on page 537	1

F: Data and Plot Names

Special Vector Data

APPENDIX G Command File Overview

This appendix presents an overview of the command file of Sentaurus Device.

[Top Level of Command File on page 1089](#) presents the topmost level of the command file. Use this table to obtain a top-down overview of the command file. The remaining sections are ordered with respect to topics. The headings of the tables, therein, mostly start with a keyword. Use these tables to find details about keywords you already know.

Organization of Command File Overview

The tables in this appendix have two or three columns, and their rows are ordered alphabetically with respect to the first (and, as far as applicable, the middle) column. Placeholders (cross references, user-supplied values in <>) precede explicit keywords, irrespective of alphabetic order.

The left column contains keywords that can appear in the command file. In the topmost rows of some tables, the left column references another table. This means that all keywords in the referenced table can appear in the referring table as well. Some keywords are followed by an opening parenthesis or an opening brace to indicate that the keyword expects a selection of options listed in the middle column of the current and the following rows.

The middle column contains options or values to the keyword in the left column, one per row. For a selection of options, the last row indicates the closing parenthesis or the closing brace. For some tables, the middle column would be empty and is omitted.

The right column contains the description for keywords, options, and values of the row. As far as applicable, in the right column, the following additional information is given:

- The default value, which can be:
 - An explicit value. Those values are written in Courier font, as you would type them.
 - ‘+’ or ‘-’ to indicate the default (true or false, respectively) for keywords that support the ‘-’ prefix.
 - ‘on’ or ‘off’ for keywords that set and reset flags, but do not support the prefix ‘-’.
 - An asterisk (*) to indicate the default choice among mutually exclusive alternatives.
 - An exclamation mark (!) if no default exists and you must specify a value.

- The unit assumed for the given quantity. In unit specifications, d denotes the dimension of the mesh. For dimensionless quantities, no unit is specified.
- The location for which keywords should be specified, given as characters in parentheses:
 - ‘g’ stands for global parameters that must not appear in region-specific, interface-specific, or contact-specific sections.
 - ‘r’ stands for region-specific (or material-specific) parameters that usually do not make sense when specified for interfaces or contacts.
 - ‘i’ and ‘c’ stand for interface-specific or contact-specific parameters.

Furthermore, the tables use the following conventions:

- An ellipsis (...) denotes that the preceding item can be repeated an arbitrary number of times.
- An asterisk (*) followed by an integer, both typeset in Times font, denotes that the preceding item must appear the given number of times.
- Optional components of specifications are enclosed in brackets and typeset in Times font.
- Angle brackets (< >) indicate user-supplied values. [Table 124](#) summarizes the common types of specification. More specific notation is explained in the last column of the table where it is used.

Some tables use additional conventions as necessary. They are explained in the text that precedes the respective table.

Table 124 Notation for user-supplied values

Specification	Description
<(x,y)> <[x,y]> <(x,y]> <[x,y)>	Range of floating-point numbers from x to y. Parentheses are used when the limits are excluded from the range; brackets are used when they are included. To denote ranges unbound on one side, the corresponding limit is omitted.
<carrier>	A carrier type, Electron or Hole.
<float>	A floating-point number (integers are special cases thereof).
<ident>	An identifier. This is like a string, but is not enclosed in double quotation marks.
<int>	An integer.
<n..m>	A range of integers, from n to m, inclusively. Either n or m can be omitted, to denote ranges that are unbound on one side.
<string>	A sequence of characters (including digits and special characters) included in double quotation marks. Unlike most Sentaurus Device input, strings are case sensitive.
<vector>	A sequence of up to three floating-point numbers, enclosed in parentheses, and separated by spaces, for example: (1.0 3 0).

Top Level of Command File

[Table 125](#) lists the top levels in the command file of Sentaurus Device.

Table 125 Top level of command file

Table 126		Specification for single devices.
Device	<ident>{ Table 126 }	Device name and device specifications (mixed mode). Device Section on page 148
File{	Table 127 }	File specification. File Section on page 156
Math{	Table 149 }	Math parameters. Math Section on page 170
Physics{	Table 159 }	Global Physics specification.
Solve{	Table 128 }	The problem to be solved.
System{	Table 144 }	The circuit (mixed mode). System Section on page 149

Device

Table 126 Device{} or single-device specification [Chapter 2 on page 47](#)

CurrentPlot{	Table 121 (Table 227)	Plot scalar data to current file. CurrentPlot Section on page 59
	Table 121 /Vector (Table 227) }	Plot vectorial data to current file. CurrentPlot Section on page 59
Electrode{	Table 146 }	Electrode specification. Electrode Section on page 50
Extraction{	<ident> [=] <ident> ...}	Process parameters. Extraction File on page 135
FarfieldPlot{	Table 228	Parameters for far-field plots (laser).
File{	Table 127	Input and output files. File Section on page 49
GainPlot{	Table 229	Parameters for gain plot (laser and LED).
GridAdaption(Table 150) {Criteria{ Table 152 }}	AGM. Chapter 32 on page 869
Math	[(Table 235)]	Math specification, potentially restricted to a location.
{	Table 149 }	

G: Command File Overview

Top Level of Command File

Table 126 Device{} or single-device specification [Chapter 2 on page 47](#)

MonteCarlo{	options}	See the <i>Sentaurus Device Monte Carlo User Guide</i> .
NoisePlot{	Table 231 }	Noise output data. Noise Output Data on page 506
NonLocalPlot	(<vector>...) { Table 232 }	Nonlocal plot. Visualizing Data Defined on Nonlocal Meshes on page 106
Physics {	[(Table 235)]	Physical models, potentially restricted to a location.
	Table 159 }	
Plot{	Table 225 }	Plot data. Plot Section on page 58
RayTraceBC{	Table 147 }	Raytrace boundary conditions (photodetector and LED).
TaileDistributionPlot{	<vector>...	Tail electron-energy distribution plot. Visualizing Tail Distribution on page 544
TailhDistributionPlot{	<vector>...	Tail hole-energy distribution plot. Visualizing Tail Distribution on page 544
TensorPlot({ 	Table 233)	Tensor plot for beam propagation method. Visualizing Results on Native Tensor Grid on page 471
	ComplexRefractiveIndex	
	OpticalField	
	OpticalIntensity}	
Thermode{	Table 148 }	Thermode specification. Thermode Section on page 53
TrappedCarDistrPlot{	<vector>	Trapped carrier charge plot at position <vector>. Visualizing Traps on page 359
	Table 238=<vector>...}}	Trapped carrier charge plot restricted to location.
VCSELNearFieldPlot{	Table 234 }	Parameters for VCSEL near-field plots (laser).

File

In the description of [Table 127 on page 1091](#), we indicate whether a file is input or output, as well as the extensions (in Courier font) that Sentaurus Device appends to the user-supplied name to obtain the full name. The tags enclosed in at-signs (@) denote the default Sentaurus Workbench variables for the particular file name. (These defaults can be changed in `tooldb.tcl`, see [Sentaurus Workbench User Guide, Global Configuration Files on page 1091](#).)

[page 205](#)). ‘(g)’ denotes keywords that must be used in global File sections only, while ‘(d)’ denotes keywords that must be used in device-specific File sections only.

Table 127 File{} File Section on page 49

ACExtract	=<string>	(g) Small-signal and noise analysis (output, _ac_des.plt, @acplot@). ACCoupled: Small-Signal AC Analysis on page 161
Bandstructure	=<string>	(d) Base name for plotting local band structure data in a laser or an LED simulation (output, _kpbandstruc_vertexX_des.plt, _kpeigenfunc_vertexX_des.plt). Plotting the Local Band Structure Data on page 838
CMIPath	=<string>	(g) Search path for compact circuit files (extension .ccf) and the corresponding shared object files (extension .so.arch). The files are parsed and added to the System section of the command file. If the environment variable STROOT_ARCH_OS_LIB is defined, the directory \$STROOT_ARCH_OS_LIB/sdevice is automatically added to CMIPath. System Section on page 149
Compressed		- Compress files.
Current	=<string>	Device currents, voltages, charges, temperatures, and times (output, _des.plt, @plot@).
CurrentCompressed		- Compress Current file.
DephasingRates	=<string>	Directory for saving dephasing rates using the second Born approximation. Computing Emission Tables on page 844 and Computing Dephasing Rates on page 847
DevFields	=<string>	(d) Space distribution for trapped carrier charge density. Must match grid file (input, .dat, .tdr). Energetic and Spatial Distribution of Traps on page 350
DevicePath	=<string>	(g) Load all files with the extension .device (or .dessis) in the directory path <string>. The directory path has the format dir1:dir2:dir3. The devices found can be used in the System section. They are not overwritten by a definition with the same name in the command file. System Section on page 149
Doping	=<string>	(d) Device doping, deep levels, mole fractions. Must match Grid file (input, .dat, .tdr, @doping@).
EmissionTable	=<string>	Tabulated emission data (output). Computing Emission Tables on page 844
EMWgrid	=<string>	(d) Tensor grid for Sentaurus Device Electromagnetic Wave Solver (EMW) (output). User-Defined Input or Tensor Grid on page 448

G: Command File Overview

Top Level of Command File

Table 127 File{} File Section on page 49

EMWinput	=<string>	(d) EMW input command file. User-Defined Input or Tensor Grid on page 448
Extraction	=<string>	(d) Extraction file (output, extraction_des.xtr). Extraction File on page 135
Gain	=<string>	(d) Modal stimulated and spontaneous emission spectra (output, _gain_des.plt), (d) Laser power spectra if specified (output, _powspec_des.dat) Plotting Gain and Power Spectra on page 855
Grid	=<string>	(d) Device geometry and mesh (input, .grd or .tdr, @grid@ or @tdr@).
GridCompressed		- Write compressed Grid and Doping files. Grid Specification on page 877
IlluminationSpectrum	=<string>	Illumination spectrum. Spectral Illumination on page 486
LifeTime	=<string>	(d) Lifetime profiles (input, .dat, .tdr). Lifetime Profiles from Files on page 311
Load	=<string>	Old simulation results (input, .sav). Plot, Save, and Load Commands on page 84
MesherInput	=<string>	Basename of boundary and command file to be used for calling Sentaurus Mesh. Specifying the Optical Solver on page 400
MobilityDoping	=<string>	(d) File from which donor and acceptor concentrations for mobility calculations are read. These will replace doping read from the Grid file for mobility calculations only. The geometry must match the Grid file (input, .tdr). Mobility Doping File on page 304
NewtonPlot	=<string>	Convergence monitoring (output). NewtonPlot on page 112
NonLocalPlot	=<string>	Data defined on nonlocal line meshes (output). Visualizing Data Defined on Nonlocal Meshes on page 106
OptFarField	=<string>	(d) Optical far field (output, _ff_<number>_des.plt for 1D, _ff_des.grd for 2D observation angle grid, _ff_Scalar_<number>_des.dat for 2D scalar far-field, _ff_Vector_<number>_des.dat for 2D vector far-field). Far Field on page 786
OptField0 to OptField9	=<string>	(d) File from which optical field data is loaded in a laser simulation. Loading Optical Modes on page 784
OpticalGenerationFile	=<string>	Load optical generation from a file. Loading Optical Generation from File on page 483

Table 127 [File{}](#) File Section on page 49

OpticalGenerationInput	=<string>	File from which optical generation rate is loaded. Loading and Saving Optical Generation from and to File on page 397
OpticalGenerationOutput	=<string>	File to which optical generation rate is written. Loading and Saving Optical Generation from and to File on page 397
OpticalSolverInput	=<string>	Command file of optical solver. Specifying the Optical Solver on page 400
Output	=<string>	"output" (g) Run-time log (output, _des.log, @log@).
Parameters	=<string>	(d) Device parameters (input, .par, @parameter@). Region-specific and Material-specific Physics on page 55
Piezo	=<string>	(d) Stress data, in Pa, for piezoelectric model (input). Chapter 23 on page 595
Plot	=<string>	Spatially distributed simulation results (output, _des.dat, _des.tdr, @dat@). Plot Section on page 58, Plot, Save, and Load Commands on page 84
PlotCompressed		- Compress Plot file.
PMIPath	=<string>	(g) Search path to load shared object files (extension .so.arch) for PMI models. Command File of Sentaurus Device on page 916
PMIUserFields	=<string>	(d) File containing any of the fields PMIUserField0 to PMIUserField9.
TaileDistribution	=<string>	Tail electron distribution versus kinetic energy (output, _des.plt). Visualizing Tail Distribution on page 544
TailhDistribution	=<string>	Tail hole distribution versus kinetic energy (output, _des.plt). Visualizing Tail Distribution on page 544
TensorPlot	=<string>	Base name for saving tensor plot files when using beam propagation method. Visualizing Results on Native Tensor Grid on page 471
Save	=<string>	Simulation results for retrieval with Load (output, _des.sav). Plot, Save, and Load Commands on page 84
SaveCompressed		- Compress Save files.
SaveOptField	=<string>	Base name for saving optical field data in a laser or an LED simulation. Saving Optical Modes on Optical or Electrical Mesh on page 783

G: Command File Overview

Top Level of Command File

Table 127 File{} File Section on page 49

SPICEPath	=<string>	Search path for SPICE circuit files (extension .scf). The files are parsed and added to the System section of the command file. If the environment variable STROOT_LIB is defined, the directory \$STROOT_LIB/sdevice/spice is automatically added to SPICEPath. SPICE Circuit Models on page 157
SponEmissionTable	=<string>	File containing tabulated spontaneous emission values to be used in a laser or an LED simulation (input). Loading Tabulated Stimulated and Spontaneous Emission on page 848
StimEmissionTable	=<string>	File containing tabulated stimulated emission values to be used in a laser or an LED simulation (input). Loading Tabulated Stimulated and Spontaneous Emission on page 848
TrappedCarPlotFile	=<string>	Trapped carrier charge density, trap occupancy probability, and trap density versus energy (output, _des.plt). Visualizing Traps on page 359

Solve

Table 128 Solve{}

Table 140		Solve selected stand-alone problem.
[<ident>.] Table 133		Solve selected equation [for instance <ident>].
ACCoupled({	Table 129)	Perform small-signal and noise analysis. ACCoupled: Small-Signal AC Analysis on page 161
	[<ident>.] Table 133... }	Equations to be solved consistently.
Continuation({	Table 130)	Continuation options.
	Table 128)	Problem to be solved.
Coupled({	Table 131)	Solve coupled equations consistently by Newton iteration.
	[<ident>.] Table 133...)	Equations to be solved.
CurrentPlot	as Load	Control current output. CurrentPlot Section on page 88

Table 128 Solve{}

HBCoupled(Table 135)	Perform harmonic balance analysis. Harmonic Balance on page 164
{	[<ident>.] Table 133...]	Equations to be solved.
Load(Table 136)	Load and continue with solution stored by Save . Plot, Save, and Load Commands on page 84
[{	Circuit	Load circuit.
<ident>...}]		Devices to be loaded.
NewCurrentPrefix	=<string>	Use prefix <string> for current file. NewCurrentPrefix Statement on page 87
Plot(Table 136)	Plot current solution. Plot, Save, and Load Commands on page 84
[{	<ident>...}]	Devices to be plotted.
Plugin(Table 138)	Solve equations consistently by Gummel iteration.
{	Table 140	Solve selected stand-alone problem.
	[<ident>.] Table 133...	Solve selected equation [for example, <ident>].
	Coupled (Table 131) { [<ident>.] Table 133... ...}	Solve coupled equations.
Quasistationary(Table 139)	Quasistationary simulation options.
{	Table 128 }	Problem to be solved.
Save	as Load	Save current solution for retrieval with Load . Plot, Save, and Load Commands on page 84

G: Command File Overview

Top Level of Command File

Table 128 Solve{}

Set	(<ident>=<float>...)	Set node. Set and Unset Section on page 169
	(<ident>. <string>= <float>...)	Set parameter <string> of compact circuit instance <ident>. Accessing SPICE Vector Parameters on page 169
	(<ident> mode Charge)	Use charge boundary condition for contact <ident>. Set Command on page 91
	(<ident> mode Current)	Use current boundary condition for contact <ident>. Set Command on page 91
	(<ident> mode Voltage)	Use voltage boundary condition for contact <ident>. Set Command on page 91
	(MSConfigs (Table 137))	Set MSCs. Manipulating MSCs During Solve on page 369
	(([Device=<string>] [MSConfig=<string>] [Transition=<string>]] PreFactor=<float>) ...)	Set MSC transition prefactors for emission, capture, or both using EPreFactor, CPreFactor, or PreFactor, respectively. The optional keys allow you to restrict the setting to specific transitions in the device-MSC-transition hierarchy. Manipulating MSCs During Solve on page 369
	(TrapFilling= Table 143)	Set trap filling. Explicit Trap Occupation on page 361
	(Traps (Table 143))	Set traps. Explicit Trap Occupation on page 361
[+]System	(<string>)	Execute UNIX command <string> [and use its return status to decide whether it was successful]. System Command on page 87
Transient(Table 141)	Transient simulation options.
{	Table 128 }	Problem to be solved.
Unset	(<ident>...)	Unset nodes. Set and Unset Section on page 169
	(TrapFilling)	Use the standard trap equations. Explicit Trap Occupation on page 361

Table 129 ACCoupled() ACCoupled: Small-Signal AC Analysis on page 161

Table 131		Coupled() parameters.
ACCompute(Table 136)	Restrict AC or noise analysis to selected points in a Quasistationary command.
ACExtract	=<string>	Prefix for the name of the file to which the results of AC analysis are written (overrides the specification from the File section).
ACMethod	=Blocked	Use Blocked solver.
ACPlot	=<string>	Plot the responses of the solution variables to the AC signals. <string> is a prefix for the names of the files to which the responses are written.
ACSubMethod	(<ident>) = Table 154	Super Inner solver for device <ident>.
CircuitNoise		Compute noise from circuit elements. Noise from SPICE Circuit Elements on page 504
Decade		Use logarithmic intervals between the frequencies.
EndFrequency	=<(0,)>	! Hz Upper frequency.
Exclude	(<ident>...)	Devices that will be removed from the circuit for AC analysis.
Extraction{	Table 15}	List of frequency-dependent extraction curves. Extraction File on page 135
Linear		Use linear intervals between the frequencies.
Node	(<ident>...)	Nodes for which to perform AC analysis.
NoisePlot	=<string>	Prefix for a file name. Chapter 16 on page 499
NumberOfPoints	=<0..>	! Number of frequencies for which to perform the analysis.
ObservationNode	(<ident>...)	Nodes for which to perform noise analysis; subset of those in Node. Chapter 16 on page 499
Optical		Perform optical AC analysis. Optical AC Analysis on page 163
PhotonicAC		Perform intensity and photon phase AC analysis. Laser Small-Signal AC Analysis and Modulation Response on page 693
PhotonicIntensityNoise		Compute relative intensity noise. Modeling Relative Intensity Noise and Frequency Noise on page 695
PhotonicPhaseNoise		Compute frequency noise. Modeling Relative Intensity Noise and Frequency Noise on page 695
StartFrequency	=<(0,)>	! Hz Lower frequency.

G: Command File Overview

Top Level of Command File

Table 130 Continuation() Continuation Command on page 75

BreakCriteria{	Table 151}	Break criteria. Break Criteria on page 99
Decrement	=<float>	1 . 5 Divisor for step size on failure to solve.
DecrementAngle	=<float>	5 deg Angle for which step starts decreasing.
Digits	=<float>	3 Number of digits for relative error.
Error	=<float>	0 . 05 Absolute error target.
Iadapt	=<float>	! Lower current limit for adaptive algorithm.
Increment	=<float>	2 Multiplier for step size on successful solve.
IncrementAngle	=<float>	2 . 5 deg Angle up to which step increases.
InitialVstep	=<float>	! Initial voltage step.
MaxCurrent	=<float>	! Upper current limit.
MaxIfactor	=<float>	! Maximum-allowed current relative to previous point as a multiplication factor.
MaxIstep	=<float>	! Maximum-allowed current step.
MaxLogIfactor		! Maximum-allowed current relative to previous point as a multiplication factor, in orders of magnitude.
MaxStep	=<float>	Maximum step for the internal arc length variable.
MaxVoltage	=<float>	! Upper voltage limit.
MaxVstep	=<float>	! Maximum-allowed voltage step.
MinCurrent	=<float>	! Lower current limit.
MinStep	=<float>	1e - 5 Minimum step for the internal arc length variable.
MinVoltage	=<float>	! Lower voltage limit.
MinVoltageStep	=<float>	1e - 2 Minimum voltage step controlling arc length step increase.
Name	=<ident>	Electrode to bias in continuation. It must be specified.
Normalized		Compute angles in a local scaled I-V coordinate system.
Rfixed	=<float>	0 . 001 Fixed resistor value.

Table 131 Coupled() [Coupled Command on page 65](#)

Digits	=<float>	Relative error target.
GridAdaption(CurrentPlot	- Plot data obtained on intermediate grids to current file. Adaptive Solve Statements on page 879
	MaxCLoops=<int>	1e5 Maximum number of adaptation iterations. Adaptive Solve Statements on page 879
	Plot)	- Plot device data on intermediate grids. Adaptive Solve Statements on page 879
IncompleteNewton (Incomplete Newton. Incomplete Newton Algorithm on page 112
	RhsFactor=<float>	Maximum change in RHS to allow old Jacobian to be reused.
	UpdateFactor=<float>)	Maximum change in update to allow old Jacobian to be reused.
Iterations	=<0...>	Maximum number of iterations of the Newton algorithm.
LineSearchDamping	=<(0,1]>	Minimal coefficient for line search damping. 1 disables damping.
Method	= Table 154	Linear solver.
	=Blocked	Block decomposition solver.
NotDamped	=<0...>	Number of Newton iterations before Bank–Rose damping is applied.
SubMethod	[(<ident>)]= Table 154	Super Inner solver [for device <ident>].

G: Command File Overview

Top Level of Command File

Table 132 Cyclic() Large-Signal Cyclic Analysis on page 81

Accuracy	=<float>	Tolerance ϵ_{cyc} .
Extrapolate(Average	+ Use averaged factor r_{av}^o for each object. Factor r is defined separately for each vertex.
	Factor=<float>	1 Coefficient f for r_{av}^o estimation.
	Forward	- Proceed as in a standard transient, without cyclic extrapolation.
	MaxVal=<float>	25 Value of r_{max} .
	MinVal=<float>	1 Value of r_{min} .
	Print	+ - Print averaged factors r_{av}^o for each object.
	QFtraps)	- Apply extrapolation to ‘trap quasi-Fermi level’ Φ_T instead of trap occupation probabilities f_T .
Period	=<float>	s Period of the cycle.
RelFactor	=<float>	Relaxation factor Υ .
StartingPeriod	=<2..>	2 Period from which the extrapolation procedure starts.

In the description, the default for ErrRef (first number; see [Table 149 on page 1112](#)) and its unit, and the default absolute error criterion (second number, see [Table 149](#)) are given.

Table 133 Equations

Circuit	0.0258 V 1e-3 Circuit equations.
CondInsulator	0.0258 V 1e-3 Conductive insulator equations.
Contact	0.0258 V 1e-3 Contact equations.
Electron	1e10 cm ⁻³ 1e-5 Electron continuity equation. Drift-Diffusion Model on page 179
eQuantumPotential	0.0258 V 1e-3 Electron quantum-potential equation. Density Gradient Quantization Model on page 260
eTemperature	300 K 1e-4 Electron temperature equation. Hydrodynamic Transport Model on page 184
Hole	1e10 cm ⁻³ 1e-5 Hole continuity equation. Drift-Diffusion Model on page 179
hQuantumPotential	0.0258 V 1e-3 Hole quantum potential equation. Density Gradient Quantization Model on page 260
hTemperature	300 K 1e-4 Hole temperature equation. Hydrodynamic Transport Model on page 184
HydrogenAtom	1e10 cm ⁻³ 1e-3 Hydrogen atom transport equation. Hydrogen Transport on page 381

Table 133 Equations

HydrogenIon	$1e10 \text{ cm}^{-3}$ $1e-3$ Hydrogen ion transport equation. Hydrogen Transport on page 381
HydrogenMolecule	$1e10 \text{ cm}^{-3}$ $1e-3$ Hydrogen molecule transport equation. Hydrogen Transport on page 381
PhotonPhase	Photon phase equation. Photon Rate and Photon Phase Equations on page 690
PhotonRate	Photon rate equation. Photon Rate and Photon Phase Equations on page 690
PhotonRecycle	Photon-recycling effect. Active-Region Photon Recycling on page 738
Poisson	0.0258 V $1e-3$ Poisson equation. Poisson Equation and Continuity Equations on page 178
SingletExciton	Singlet exciton equation. Singlet Exciton Equation on page 189
TCircuit	Thermal circuit equations.
TContact	Thermal contact equations.
Temperature	300 K $1e-3$ Temperature equation. Thermodynamic Transport Model on page 179 , Hydrodynamic Transport Model on page 184

Table 134 Goal{} Quasistationary Command on page 67, Quasistationary Command on page 160

Charge	=<float>	C Target charge for contact.
Contact	=<string1>.<string2>	Name of instance and contact to be ramped.
Current	=<float>	Target current for contact.
DopingWell	(<vector>)	Semiconductor well defined by point <vector> where quasi-Fermi potential will be ramped.
DopingWells	(Region=<string>)	Semiconductor wells in the region <string> where quasi-Fermi potential will be ramped.
	(Material=<string>)	Semiconductor wells in material <string> where quasi-Fermi potential will be ramped.
	(Semiconductor)	All device semiconductor wells where Fermi potential will be ramped.
eQuasiFermi	=<float>	V Target electron quasi-Fermi potential for semiconductor wells. Ramping Quasi-Fermi Potentials in Doping Wells on page 69
hQuasiFermi	=<float>	V Target hole quasi-Fermi potential for semiconductor wells. Ramping Quasi-Fermi Potentials in Doping Wells on page 69

G: Command File Overview

Top Level of Command File

Table 134 Goal{} Quasistationary Command on page 67, Quasistationary Command on page 160

Model	=<string>	Name of model for which the parameter will be ramped.
ModelProperty	=<string>	Parameter path or name of parameter that is ramped when using Unified Interface for Optical Generation Computation on page 393 .
	=<string1>.<string2>	Name of instance and parameter path or name of parameter that is ramped when using Unified Interface for Optical Generation Computation on page 393 .
Name	=<string>	Name of contact to be ramped.
Node	=<string>	Name of node for which the voltage will be ramped.
Parameter	=<string>	Name of parameter that will be ramped.
	=<string1>.<string2>	Name of instance and parameter that will be ramped.
Power	=<float>	Target heat for contact.
Temperature	=<float>	K Target temperature for contact.
Value	=<float>	Target value for parameter.
Voltage	=<float>	V Target voltage for node or contact.
WellContactName	=<string>	Name of contact defining the well where quasi-Fermi potential will be ramped.

Table 135 HBCoupled() Harmonic Balance on page 164

Table 131		Coupled() parameters.
CNormPrint		+ Print instance equation errors per Newton step (MDFT mode only).
Derivative		- Use complete Jacobian for HB Newton.
GMRES(Use GMRES linear solver; requires Method=ILS.
	MaxIterations=<int>	200 Maximal number of iterations.
	Restart=<int>	20 Size of minimization subspace.
	Tolerance=<float>	1e-4 Required residuum reduction.
Initialize	=DCMode	0-th harmonic from DC solution; others, zero.
	=HBMode	All harmonics from previous harmonic balance (HB) solution.
	=MixedMode	0-th harmonic from DC; others, from previous HB solution.

Table 135 HBCoupled() Harmonic Balance on page 164

Method	=ILS	Required for GMRES.
	=Pardiso	Use PARDISO to solve linear system (SDFT mode only).
Name	=<string>	Name component of plot files. Default is Coupled_<number>, where <number> is the global index of all Coupled, ACCoupled, and HBCoupled solve entries.
RhsScale	(Table 133)=<float>	Scaling of RHS in Newton (MDFT mode only).
SolveSpectrum	=<string>	Referenced solve spectrum (MDFT mode only).
Tone(Multiple specification for multitone (MDFT mode only).
	Frequency=<freqspec>	! Hz Base frequency. Performing Harmonic Balance Analysis on page 165
	NumberOfHarmonics=<int>	! Number of harmonics H . Eq. 23, p. 164
UpdateScale	(Table 133)=<float>	Scaling of update in Newton (MDFT mode only).
ValueMin	(Table 133)=<float>	Lower bound for quantity in time domain (MDFT mode only).
ValueVariation	(Table 133)=<float>	Allowed variation of quantity in time domain (MDFT mode only).

Table 136 Load(), Plot(), Save() in Solve{} Plot, Save, and Load Commands on page 84

Compressed		off Write compressed files. Do not use if Sentaurus Device runs under Sentaurus Workbench.
Current	(Difference=<float>)	A Analogous to Voltage.
	(Intervals=<int>)	Analogous to Voltage. When LogCurrent is given in Continuation, use logarithmic current range.
	(LogDifference=<(0,)>)	Analogous to Voltage. Split interval into logarithmic intervals of size <float>.
Explicit		- Write datasets specified in Plot section only.
FilePrefix	=<string>	Prefix of file name. File names consist of the prefix, the instance name, an optional local number (depending on Overwrite and noOverwrite), and an extension. The default file prefix is save<globalsaveindex> for Save and plot<globalplotindex> for Plot.
Iterations	=(<int>;...)	(0;1;...) Save or plot at certain Plugin iterations.
IterationStep	=<int>	Save or plot all <int> steps of plugins, quasistationaries, continuations, or transients.

G: Command File Overview

Top Level of Command File

[Table 136 Load\(\), Plot\(\), Save\(\) in Solve{} Plot, Save, and Load Commands on page 84](#)

Loadable		+ Write additional information required to load a simulation from a .tdr file.
noOverwrite		off Give each new save or plot file a new name by numbering.
Number	=<int>	Number of the solution to be retrieved by Load.
Overwrite		on Rewrite the same file name at each loop.
Time	=(<float>;...)	List of times when data is saved or plotted.
unCompressed		on Write uncompressed files.
Voltage	(Difference=<float>)	V Only for Continuation simulations. Divide the range specified with MinVoltage and MaxVoltage of Continuation into intervals of size <float>, and write save or plot files every time the voltage enters one of these intervals.
	(Intervals=<int>)	Only for Continuation simulations. Divide the range specified with MinVoltage and MaxVoltage of Continuation into <int> intervals, and write save or plot files every time the voltage enters one of these intervals.
When(Generate output whenever the target was crossed between the previous and the current iteration i , $X_{i-1} < X_T \leq X_i$ or $X_{i-1} > X_T \geq X_i$. Available for Plot, Save, and CurrentPlot inside a Quasistationary, Transient, or Continuation.
Contact=[<string>.]<string>		[Instance and] contact name for target. The instance name defaults to "", appropriate for single-device simulation.
Current=<float>	A	Target current X_T .
Node=<string>		Node name for target.
Voltage=<float>)	V	Target voltage X_T .

[Table 137 MSConfigs\(\) in Set\(\) in Solve{} Manipulating MSCs During Solve on page 369](#)

Frozen		+ Freeze MSCs.
MSConfig(Set occupations for the specified MSC.
	Device=<string>	Device instance name to where the MSC belongs.
	Name=<string>	Name of MSC.
	State(Name=<string> Value=<float>)...)	Set occupation of MSC state <string> to given value.

Table 138 [Plugin\(\) Plugin Command on page 66](#)

BreakOnFailure		Stop when an inner Coupled fails.
Digits	=<float>	Relative precision target.
Iterations	=<int>	Maximum number of iterations. 0 is used to perform one loop without error testing.

Table 139 [Quasistationary\(\) Quasistationary Command on page 67](#), [Quasistationary Command on page 160](#)

BreakCriteria{	Table 151	Break criteria. Break Criteria on page 99
Decrement	=<float>	2 Divisor for the step size when last step failed.
DoZero		+ - The equations are solved for $t = 0$.
Extraction{	Table 15	List of voltage-dependent extraction curves. Extraction File on page 135
Goal{	Table 134	Goal for ramping.
GridAdaption(CurrentPlot	- Write currents obtained on intermediate grids. Adaptive Solve Statements on page 879
	Iterations=<int>;...)	Adapt grid at given iterations. Adaptive Quasistationary Solve Statements on page 880
	IterationStep=<int>	Adapt grid any <int> steps. Adaptive Quasistationary Solve Statements on page 880
	MaxCLoops=<int>	1e5 Maximum number of adaptation iterations. Adaptive Solve Statements on page 879
	Plot	- Plot device data on intermediate grids. Adaptive Solve Statements on page 879
	Time= (Range=<int>*2) ;...)	Adapt grid when time falls into a given range. Adaptive Quasistationary Solve Statements on page 880
Increment	=<float>	2 Multiplier for the step size when last step was successful.
InitialStep	=<float>	0.1 Initial step size.
MaxStep	=<float>	1 Maximum step size.
MinStep	=<float>	0.001 Minimum step size.

G: Command File Overview

Top Level of Command File

Table 139 Quasistationary() [Quasistationary Command on page 67](#), [Quasistationary Command on page 160](#)

NewtonPlotStep	=<float>	Upper limit for step size for which to write Newton plot files. Saving Snapshots on page 134
Plot{	Intervals=<int>	! Number of intervals in Range. Saving and Plotting Data During Quasistationary Solve Sequence on page 73
	Range=(<float>*2) }	! Range of t for which to generate plots. Saving and Plotting Data During Quasistationary Solve Sequence on page 73
PlotBandstructure	as Plot	Plot band structure data in a laser or an LED simulation. Plotting the Local Band Structure Data on page 838
PlotFarField	as Plot	Optical far field. Far Field on page 786
PlotGain	as Plot	Plot stimulated and spontaneous emission data in laser and LED simulations. Modal Gain as Function of Energy/Wavelength on page 856
ReadExtrapolation		+ Try to use the extrapolation information from a previous Quasistationary if it is available and compatible. Extrapolation on page 73
SaveOptField	as Plot	Save optical field data in a laser or an LED simulation. Saving Optical Modes on Optical or Electrical Mesh on page 783
StoreExtrapolation		+ Store the extrapolation information internally at the end of the Quasistationary, so that it is available for a subsequent Quasistationary or can be written to a save or plot file. Extrapolation on page 73

Table 140 Standalone

Bandstructure	Band structure using $k \cdot p$ method. The k.p Method on page 825
DephasingRates	Dephasing rates for tabulated optical emission using second Born approximation. Computing Tabulated Optical Emission Data on page 844
EmissionTable	Tabulated stimulated and spontaneous emission. Computing Tabulated Optical Emission Data on page 844
Optics	Optical problem.
Wavelength	Update wavelength according to optical problem.

Table 141 Transient() Transient Command on page 79

Table 157		Time step control.
Cyclic(Table 132)	Cyclic analysis. Large-Signal Cyclic Analysis on page 81
FinalTime	=<float>	s Final time.
InitialStep	=<float>	0 . 1 s Initial step size.
InitialTime	=<float>	0 s Start time.
MaxStep	=<float>	1 s Maximum step size.
MinStep	=<float>	0 . 001 s Minimum step size.
NewtonPlotStep	=<float>	Upper limit for step size for which to write Newton plot files. Saving Snapshots on page 134
Plot{	Intervals=<int>	Number of intervals in Range. Saving and Plotting Data During Quasistationary Solve Sequence on page 73
	Range=(<float>*2) }	s Range of t for which to generate plots. Saving and Plotting Data During Quasistationary Solve Sequence on page 73
ReadExtrapolation		+ Try to use the extrapolation information from a previous Transient if it is available and compatible. Extrapolation on page 80
StoreExtrapolation		+ Store the extrapolation information internally at the end of the Transient, so that it is available for a subsequent Transient or can be written to a save or plot file. Extrapolation on page 80

G: Command File Overview

Top Level of Command File

Table 142 TrapFilling= in Set() in Solve {} [Explicit Trap Occupation on page 361](#)

0	Set trap occupation to be in equilibrium with zero electron and hole concentration. Explicit Trap Occupation on page 361
-Degradation	Return trap concentrations to initial values. Explicit Trap Occupation on page 361 , Device Lifetime and Simulation on page 378
Empty	Set all traps to empty. Explicit Trap Occupation on page 361
Frozen	Keep the current trap occupation unchanged until the next Set or UnSet. Explicit Trap Occupation on page 361
Full	Set all traps to fully occupied. Explicit Trap Occupation on page 361
n	Set trap occupation to be in equilibrium with a very high electron and zero hole concentration. Explicit Trap Occupation on page 361
p	Set trap occupation to be in equilibrium with a very high hole and zero electron concentration. Explicit Trap Occupation on page 361

Table 143 Traps() in Set() in Solve {} [Explicit Trap Occupation on page 361](#)

[<string1>.]<string2>=<float>...	Set occupation of trap <string2> of device <string1> to specified value. Explicit Trap Occupation on page 361
Frozen	+ Freeze traps. Explicit Trap Occupation on page 361

System

Table 144 System{} [System Section on page 149](#)

<ident1>	<ident2>(<string>=<ident3>...)	Create device instance <ident2> from device <ident1> and connect electrode <string> to node <ident3>.
<ident1>	<ident2>(<ident3>...) {<ident4>=<pvalue>...}	Create instance <ident2> from parameter set <ident1>, connect terminals to nodes <ident3>, and override parameter <ident4> by <pvalue>, the type of which depends on the parameter.
ACPlot(Table 145	Define circuit quantities for AC analysis output. AC System Plot on page 155
Electrical	(<ident>...)	List of electrical nodes. System Section on page 149
HBPlot	[<string>] (Table 145)	Define circuit quantities for HB analysis output. Harmonic Balance on page 164

Table 144 System{} System Section on page 149

Hint(<ident>=<float>)	Set node only for the first solve. Set, Unset, Initialize, and Hint on page 154
Initialize(<ident>=<float>)	Set node until first transient. Set, Unset, Initialize, and Hint on page 154.
Plot	[<string>] (Table 145)	Plot circuit quantities to file named <string>.
Set(<ident>=<float>	Set node. Set, Unset, Initialize, and Hint on page 154
	<ident>.<string>=<float>	Set parameter <string> of compact circuit instance <ident>.
Thermal	(<ident> ...)	List of thermal nodes. System Section on page 149
Unset(<ident>)	Unset node. Set, Unset, Initialize, and Hint on page 154

Table 145 Plot(), ACPlot(), and HBPlot() in System{} System Plot on page 155, AC System Plot on page 155, Harmonic Balance on page 164

<ident>		Print the voltage at node <ident>.
freq	()	Print the current AC analysis frequency.
h	(<ident1> <ident2>)	Print the heat that exits device <ident1> through node <ident2>.
i	(<ident1> <ident2>)	Print the current that exits device <ident1> through node <ident2>.
p	(<ident1> <ident2>)	Print attribute <ident2> of circuit element <ident1>.
t	(<ident>)	Print the temperature at node <ident>.
	(<ident1> <ident2>)	Print the temperature difference between two given nodes.
time	()	Print the current time in transient analysis, or quasistationary t parameter for frequency-domain analysis.
v	(<ident>)	Print the voltage at node <ident>.
	(<ident1> <ident2>)	Print the voltage difference between two given nodes.

Boundary Conditions

Table 146 Electrode{} Electrode Section on page 50, Electrical Boundary Conditions on page 206

AreaFactor	=<float>	1 Multiplier for electrode current. Electrode Section on page 50
Barrier	=<float>	V Barrier voltage.
Charge	=<float>	C Charge for floating electrode. Voltage must not be specified. Floating Semiconductor Gates on page 213
Current	=<float>	A μ m ^{d-3} Current boundary condition. Voltage is used as initial guess only.
DistResist	=<float>	Ω cm ² Distributed resistance. Resistive Contacts on page 210
	=SchottkyResist	Use Schottky contact resistance model. Resistive Contacts on page 210
eRecVelocity	=<[0,)>	2.573e6 cms ⁻¹ Electron recombination velocity. Schottky Contacts on page 208
Extraction{	bulk	Specify electrode as bulk for extraction purposes. Extraction File on page 135
	drain	Specify electrode as drain for extraction purposes. Extraction File on page 135
	gate	Specify electrode as gate for extraction purposes. Extraction File on page 135
	source}	Specify electrode as source for extraction purposes. Extraction File on page 135
FGcap=(value=<float>	F μ m ^{d-3} Additional capacitance for floating electrode. Floating Metal Gates on page 211
	name=<string>)	Coupling to electrode <string>. Floating Metal Gates on page 211
hRecVelocity	=<[0,)>	1.93e6 cms ⁻¹ Hole recombination velocity. Schottky Contacts on page 208
Material	=<string>	Electrode material. Gate Contacts on page 207
	=<string> (N=<(0,)>)	cm ⁻³ Electrode material with n-doping. Gate Contacts on page 207
	=<string> (P=<(0,)>)	cm ⁻³ Electrode material with p-doping. Gate Contacts on page 207

Table 146 Electrode{} [Electrode Section on page 50](#), [Electrical Boundary Conditions on page 206](#)

Resist	=<float>	$\Omega \mu\text{m}^{3-d}$ Contact resistance. Resistive Contacts on page 210
Schottky		off Contact is a Schottky contact. Schottky Contacts on page 208
Voltage	=<float>	V Contact voltage.
Workfunction	=<float>	eV Electrode workfunction. Gate Contacts on page 207

Table 147 RayTraceBC{} [Boundary Condition for Raytracing on page 422](#), [Setting Up Simple Photon-Recycling Model on page 740](#), [Syntax for Full Photon-Recycling Model on page 746](#)

FullPhotonRecycle	=ReEmit(<float>)	0 Re-emission coefficient.
Name	=<string>	Name of the reflectivity contact or photon-recycling contact.
LayerStructure{	<float> <string>; <float> <string>; ... <float> <string>}	Definition of multilayer structure used for TMM calculation. First column contains thickness of layer in μm . Second column contains material name of layer.
PhotonRecycle	=AbsEmit(<float>*2)	0 0 Absorption and re-emission coefficients.
	=ASE(<float>)	1 Amplified spontaneous emission factor.
pmiModel	=<string>	Name of the PMI model associated with this BC contact.
ReferenceMaterial	=<string>	Definition of LayerStructure orientation. The topmost layer in the LayerStructure specification is connected to the region with material ReferenceMaterial.
ReferenceRegion	=<string>	Definition of LayerStructure orientation. The topmost layer in the LayerStructure specification is connected to the region with name ReferenceRegion.
Reflectivity	=<[0,1]>	0 Reflectivity.
Transmittivity	=<[0,1]>	0 Transmittivity.

Table 148 Thermode{} [Thermal Boundary Conditions for Thermodynamic Model on page 215](#)

AreaFactor	=<float>	1 Multiplier for heat fluxes. Electrode Section on page 50
Power	=<float>	Wcm^{-2} Heat flux boundary condition. An additional Temperature specification serves as initial guess only.
SurfaceConductance	=<float>	$\text{cm}^{-2}\text{K}^{-1}\text{W}$ Contact thermal conductance.

G: Command File Overview

Math

Table 148 `Thermode{}` Thermal Boundary Conditions for Thermodynamic Model on page 215

SurfaceResistance	=<float>	cm ² KW ⁻¹ Contact thermal resistivity.
Temperature	=<float>	K Contact temperature.

Math

Table 149 `Math{}` Math Section on page 91, Math Section on page 170

Table 157		Transient time-step control.
ACMethod	=Blocked	* Use block decomposition solver for ACCoupled.
ACSubMethod	= Table 154	Inner linear solver for Blocked method for ACCoupled.
AutomaticCircuitContact		on Poisson covers the circuit. Circuit and Contact Equation–Variable Keywords on page 158
AvalDerivatives		+ Compute analytic derivatives of avalanche generation. Derivatives on page 93
AverageBoxMethod		+ Use element-oriented element-intersection BM algorithm. If disabled, use quadrilateral BM algorithm. Box Method Coefficients on page 886
BoxCoefficientsFromFile	[(GrdNumbering)]	Try to read sections of the geometry file. Saving and Restoring Box Method Coefficients on page 891
BoxMeasureFromFile	[(GrdNumbering)]	Try to read sections of the geometry file. Saving and Restoring Box Method Coefficients on page 891
BoxMethodFromFile		+ Read Voronoï surface from this file if the grid file has a <code>VoronoiFaces</code> section. Box Method Coefficients on page 886
BreakAtIonIntegral	(<int> <float>)	1 1 Terminate the quasistationary simulation when the <int> largest ionization integrals are greater than <float>. Approximate Breakdown Analysis: Poisson Equation Approach on page 333
BreakCriteria{	Table 151 }	Break criteria. Break Criteria on page 99

Table 149 Math{} Math Section on page 91, Math Section on page 170

CheckUndefinedModels		+ (g) Check for undefined physical parameters. Undefined Physical Models on page 119
CNormPrint		(g) Convergence monitoring. CNormPrint on page 111
ComputeIonizationIntegrals		off Compute ionization integrals for paths that cross local field maxima in a semiconductor. Approximate Breakdown Analysis: Poisson Equation Approach on page 333
	(WriteAll)	off Output information for all computed paths.
ConstRefPot	=<float>	eV Value for ψ_{ref} . Discretization Methods on page 93
cT_Range	=(<float*2>)	10 80000 K (g) Lower and upper limit for carrier temperature. Nonlinear Solver-oriented Math Keywords on page 97
CurrentWeighting		off Compute contact currents using an optimal weighting scheme. Discretization Methods on page 93
Cylindrical	(<float>)	off Use the 2D mesh to simulate a 3D cylindrical device. The device is assumed to be rotationally symmetric around the vertical axis given by $x = <float>\mu\text{m}$. $<float>$ must be less than or equal to the smallest horizontal device coordinate, and defaults to 0. Physics-related Math Keywords on page 92
DensLowLimit	=<float>	1e-100 cm ⁻³ (g) Lower limit for carrier densities. Nonlinear Solver-oriented Math Keywords on page 97
Derivatives		+ Compute analytic derivatives of mobility and avalanche generation. Derivatives on page 93
Digits	=<float>	5 Approximate number of digits to which equations must be solved to be considered as converged. Eq. 12, Eq. 14, Nonlinear Solver-oriented Math Keywords on page 97
	(NonLocal) =<float>	2 (ci) Accuracy for nonlocal tunneling currents. Nonlocal Tunneling Parameters on page 521
DirectCurrent		off Compute contact currents directly, using only contact nodes and their neighbors. Discretization Methods on page 93

G: Command File Overview

Math

Table 149 Math{} Math Section on page 91, Math Section on page 170

eDrForceRefDens	=< [0,) >	0 cm^{-3} (r) Damping parameter for high-field mobility driving force. Driving Force Models on page 300
eMobilityAveraging	=Element	* (r) Use element averaged electron mobility. Mobility Averaging on page 304
	=ElementEdge	(r) Use element-edge averaged electron mobility. Mobility Averaging on page 304
EnergyResolution	(NonLocal) =< (0,) >	0 . 005 eV (ci) Minimum energy resolution for integrals in computation of nonlocal tunneling current. Nonlocal Tunneling Parameters on page 521
EnormalInterface(MaterialInterface= [<string>...]	Material interfaces for normal electric field computation. Discretization Methods on page 93
	RegionInterface= [<string>...])	Region interfaces for normal electric field computation. Discretization Methods on page 93
EpsCharge	=< [0,) >	(g) 1e -20 C Small floating-gate charge; smaller charge will not be resolved by time step.
EquilibriumSolution(Table 131)	(g) Parameters for equilibrium solution used with conductive insulators. Conductive Insulators on page 195
Error	(Table 133) =<float>	Value of ε_A . Eq. 12, Nonlinear Solver-oriented Math Keywords on page 97
ErrRef	(Table 133) =<float>	Value of x_{ref} . Eq. 14, Nonlinear Solver-oriented Math Keywords on page 97
EvEpara		+ Use accurate and reliable (but somewhat expensive) computational approach for models that depend on the parallel electric field. Discretization Methods on page 93
ExitOnFailure		(g) off Terminate the simulation as soon as a Solve command fails.
ExitOnUnknownParameterRegion		+ (g) Exit when unknown region, region interface, or electrode appears in parameter file.
ExtendedPrecision	[(Table 8)]	(g) off Use extended precision floating-point arithmetic. Extended Precision on page 113
Extrapolate		off Obtain the initial guess for the solution of the current step by extrapolating the solutions of the previous two steps. Keywords for Transient and Quasistationary Control on page 99

Table 149 Math{} Math Section on page 91, Math Section on page 170

GeometricDistances		- (g) Use enhanced distance and normal to interface computation for mobility and MLDA. Normal to Interface on page 286
HB	{ Table 153 }	Harmonic Balance on page 164 ; not device specific.
hDrForceRefDens	=< [0,) >	0 cm ⁻³ (r) Damping parameter for high-field mobility driving force. Driving Force Models on page 300
hMobilityAveraging	=Element	* (r) Use element averaged hole mobility. Mobility Averaging on page 304
	=ElementEdge	(r) Use element-edge averaged hole mobility. Mobility Averaging on page 304
IgnoreTdrUnits		off (g) Ignore TDR units when loading data from a .tdr file. Plot, Save, and Load Commands on page 84
ILSrc	=<string>	ILS options. Linear Solver-oriented Math Keywords on page 95
(- (g) Incomplete Newton. Incomplete Newton Algorithm on page 112
	RhsFactor=<float>	1.0 Maximum change in RHS to allow old Jacobian to be reused.
	UpdateFactor=<float>	0.1 Maximum change in update to allow old Jacobian to be reused.
Interrupt	=BreakRequest	Write .tdr or .sav file and abort actual solve statement after signal INT occurs. Saving Snapshots on page 134
	=PlotRequest	Write .tdr or .sav file and continue simulation after signal INT occurs. Saving Snapshots on page 134
Iterations	=<0...>	50 Maximum number of Newton iterations. If the equation being solved converges quadratically, the number of iterations can become larger. For Iterations=0, only one iteration is performed. Nonlinear Solver-oriented Math Keywords on page 97
LineSearchDamping	=<(0,1]>	1 Smallest allowed damping coefficient for line search damping. Nonlinear Solver-oriented Math Keywords on page 97
lT_Range	=(<float*2)	50 5000 K (g) Lower and upper limit for lattice temperature. Nonlinear Solver-oriented Math Keywords on page 97

G: Command File Overview

Math

Table 149 Math{} Math Section on page 91, Math Section on page 170

MetalConductivity		+ Conductivity of metals. The thermal conductivity is simulated in any case. Conductivity of Metals on page 193
Method	= Table 154	Linear solver for Coupled.
	=Blocked	* Use block decomposition solver for Coupled. Linear Solver-oriented Math Keywords on page 95
MLDAbox ({	Table 155 }...)	A box within which the interface is confined for the calculation of the MLDA distance function. Modified Local-Density Approximation on page 264
NaturalBoxMethod		- Use edge-oriented element intersection BM algorithm. Box Method Coefficients on page 886
NewtonPlot ((g) Convergence monitoring. NewtonPlot on page 112
	Error	Write the error of all solution variables.
	MinError	Write file only if error decreases.
	NewtonPlotStep=<float>	Upper limit for step size for which to write files.
	Plot	Write everything specified in the Plot section. Plot Section on page 58
	Residual	Write the residuals (right-hand sides) of all equations.
	Update)	Write the updates of all solution variables from the previous step.
NewWavelengthSearch	[(InitialWavelength =<float>)]	More robust wavelength search algorithm for laser. See note at end of Adjusting k,p Parameters in Parameter File on page 836 .
NoAutomaticCircuitContact		off (g) Poisson excludes the circuit. Circuit and Contact Equation–Variable Keywords on page 158
NonLocal	(Table 156)	(cir) Nonlocal mesh. Nonlocal Meshes on page 104
	<string> (Table 156)	(g) Named nonlocal mesh. Nonlocal Meshes on page 104

Table 149 Math{} Math Section on page 91, Math Section on page 170

NormalFieldCorrection	=<float>	(r) Normal field correction factor for mobility on interface points. Field Correction on Interface on page 287
NoSRHperPotential		off Omit potential derivatives of SRH recombination rate. Using Field Enhancement on page 314
NoSRHperT		off Omit temperature derivatives of SRH recombination rate. Using Field Enhancement on page 314
NotDamped	=<0...>	1 0 0 0 (g) Number of iterations in each Newton iteration before Bank–Rose damping is activated. Nonlinear Solver-oriented Math Keywords on page 97
Number_of_Assembly_Threads	=<int>	1 (g) Number of threads for linear solver. Parallelization on page 102
Number_of_Solver_Threads	=<int>	1 (g) Number of threads for assembly. Parallelization on page 102
Number_of_Threads	=<int>	1 (g) Number of threads for linear solver and assembly. Parallelization on page 102
Numerically	[(Table 133...)]	off (g) Use numeric derivatives. The optional equation list restricts the numeric computation to specific Jacobian blocks. Does not work with the method Blocked and, generally, is discouraged. Derivatives on page 93
ParallelToInterfaceInBoundaryLayer		+ (r) Controls the computation of driving forces for mobility and avalanche models along interfaces. Physics-related Math Keywords on page 92
(PartialLayer	* Apply switch only to elements that touch an interface by an edge (2D) or a face (in 3D).
	FullLayer)	Apply switch to all elements that touch an interface either by a face, an edge, or a vertex.

G: Command File Overview

Math

Table 149 Math{} Math Section on page 91, Math Section on page 170

PeriodicBC()		(g) Periodic boundary conditions. Periodic Boundary Conditions on page 217
	Table 133	Equation for which to apply PBCs. If omitted, apply to all equations.
	Coordinates=(<float>*2)	μm Coordinate of Direction axis where the PBCs will be applied. Sentaurus Device replaces coordinates outside the device with coordinates at the outer boundary.
	Direction=<0..2>...)	Direction of periodicity: 0 for x -axis, 1 for y -axis, and 2 for z -axis.
PlotExplicit		- (g) All Plot statements write datasets specified in Plot section only.
PlotLoadable		+ (g) All Plot statements write additional information required to load a simulation from a.tdr file.
PostProcess	(Transient=<string>)	(g) Use PMI interface <string> to postprocess data. Postprocess for Transient Simulation on page 1031
RecBoxIntegr	(<float> <int> <int>)	(1e-2 10 1000) Maximum relative deviation of covered volume, maximum number of levels, maximum number of total rectangles per box. Using Optical Beam Absorption on page 436
RecomputeQFP		Keep density variables constant, and recompute quasi-Fermi potentials when the electrostatic potential changes and carrier equations are not solved. Physics-related Math Keywords on page 92
RelErrControl		+ Use the unscaled expression Eq. 14, p. 98 for error control. Nonlinear Solver-oriented Math Keywords on page 97
RelTermMinDensity	=<(0,)>	1e3 cm ⁻³ (g) Stabilization parameter for temperature relaxation term. Hydrodynamic Model Parameters on page 188
RelTermMinDensityZero	=<(0,)>	2e8 cm ⁻³ (g) Stabilization parameter for temperature relaxation term. Hydrodynamic Model Parameters on page 188
RhsFactor	=<float>	1e10 Maximum increase of the L_2 -norm of the RHS between Newton iterations. Nonlinear Solver-oriented Math Keywords on page 97

Table 149 Math{} Math Section on page 91, Math Section on page 170

RhsMax	=<float>	1e15 Maximum of L_2 -norm of the RHS in each Newton iteration. Used only during transient simulations. Nonlinear Solver-oriented Math Keywords on page 97
RhsMin	=<float>	1e-5 Minimum of L_2 -norm of the RHS in each Newton iteration. Nonlinear Solver-oriented Math Keywords on page 97
Smooth		off Keep mobility and recombination rates from the previous step to obtain better initial conditions for extreme nonlinear iterations.
Spice_gmin	=<float>	1e-12 S (g) SPICE minimum conductance. Math Section on page 170
Spice_Temperature	=<float>	300.15 K (g) Temperature of SPICE circuit. Math Section on page 170
StackSize	=<int>	1000000 byte (g) Stack size per thread. Parallelization on page 102
StressMobilityDependence	=TensorFactor	Stress Tensor Applied to Low-Field Mobility on page 634
SubMethod	= Table 154	(g) Inner linear solver for Blocked method. Linear Solver-oriented Math Keywords on page 95
TailDistributionIterations	=<0...>	20 (g) Number of iterations in the tail distribution model. Using Tail Distribution Hot-Carrier Injection Model on page 541
TailDistributionMethod	= Table 154	super (g) Linear solver for the tail distribution model. Using Tail Distribution Hot-Carrier Injection Model on page 541
TailDistributionSORParameter	=<[1,2)>	1.1 (g) Successive over-relaxation parameter in the tail distribution model. Using Tail Distribution Hot-Carrier Injection Model on page 541
TensorGridAniso		off (g) Use tensor-grid approximation for anisotropic mobility. Tensor Grid Option on page 630
Transient	=BE	(g) Backward Euler method. Backward Euler Method on page 901
	=TRBDF	* (g) TRBDF method. TRBDF Composite Method on page 902

G: Command File Overview

Math

Table 149 Math{} Math Section on page 91, Math Section on page 170

Traps(Damping=<[0,)>	10 Damping for traps for the nonlinear Poisson equation. A value of 0 disables damping. Chapter 10 on page 349
	RegionWiseAssembly)	- Apply regionwise assembly for traps.
VoronoiFaceBoxMethod	= Table 158	Use element-face intersection box method algorithm. Box Method Coefficients on page 886
Wallclock		- (g) Report wallclock times rather than CPU times after each step in the simulation.

Table 150 GridAdaption() [Chapter 32 on page 869](#)

MaxCLoops	=<int>	1e5 Maximal number of iterations per adaptive coupled system.
MaxNumberMacroElements	=<int>	1e5 Maximal number of leaf elements in macro element tree.
NeighborSizeRatio	(Conductor)=<float>	1e6 Allowed anisotropic edge length ratios for neighboring elements in conductors.
	(Insulator)=<float>	1e6 Allowed anisotropic edge length ratios for neighboring elements in insulators.
	(Semiconductor)=<float>	3 Allowed anisotropic edge length ratios for neighboring elements in semiconductors.
NeighborSizeRefinement		+ Perform neighbor size refinement.
PatchMode	=Element	* For Dirichlet problems, compute local reference solution on $h/2$ -grid.
	=Global	For Dirichlet problems, compute global reference solution on $h/2$ -grid.
Poisson		+ Perform EPC smoothing step.
RCLoop{	Coarsening	+ Allow coarsening adaptation.
	Iterations=<int>}	2 Maximum iterations on element tree.
Smooth		+ Perform NBFI smoothing step.

Table 150 GridAdaption() [Chapter 32 on page 869](#)

Weights(Avalanche=<float>	1 Avalanche weight in AGM dissipation rate.
	eCurrent=<float>	1 Electron current weight in AGM dissipation rate.
	hCurrent=<float>	1 Hole current weight in AGM dissipation rate.
	Recombination=<float>)	1 Recombination weight in AGM dissipation rate.

Table 151 BreakCriteria{} [Break Criteria on page 99](#)

Current(absval=<float>	A μm^{d-3} Upper limit for absolute current value.
	contact=<string>	! Name of contact with break criterion.
	DevName=<ident>	Identifier of circuit device name (mixed mode only).
	maxval=<float>	A μm^{d-3} Upper limit for current.
	minval=<float>	A μm^{d-3} Lower limit for current.
	Node=<ident>)	Identifier of circuit node name (mixed mode only).
CurrentDensity(DevName=<ident>	Identifier of device (mixed mode only).
	maxval=<float>)	A cm^{-2} (r) Maximum current density.
DevicePower(absval=<float>	W μm^{d-3} Upper limit for absolute power value.
	DevName=<ident>	Identifier of circuit device name (mixed mode only).
	maxval=<float>	W μm^{d-3} Upper limit for power value.
	minval=<float>)	W μm^{d-3} Lower limit for power value.
ElectricField	DevName=<ident>	Identifier of circuit device name (mixed mode only).
	(maxval=<float>)	V cm^{-1} (r) Maximum electric field.
LatticeTemperature	DevName=<ident>	Identifier of circuit device name (mixed mode only).
	(maxval=<float>)	K (r) Maximum lattice temperature.
Voltage	as Current	V Voltage break criteria.

G: Command File Overview

Math

Table 152 Criteria{} [Adaptation Criteria on page 878](#)

Dirichlet(AbsError=<float>	Absolute error target.
	DataName="AGMDissipationRate"	Use dissipation rate as error criterion. Criteria Based on Local Dirichlet Problems on page 872
	DataName="DomainIntegralCurrent"	Use domain integral current as error criterion. Criteria Based on Local Dirichlet Problems on page 872
	RelError=<float>)	Relative error target.
Element(DataName=" Table 121 "	Dataset for criterion (must be defined on vertices). Criteria Based on Element Variation on page 873
	MaxTransDiff=<float>)	Accuracy target.
Residual(AbsError=<float>	Absolute error target.
	DataName="AGMDissipationRate"	Use dissipation rate as error criterion. Residual Adaptation Criteria on page 873
	RelError=<float>)	Relative error target.

Table 153 HB{} [Harmonic Balance on page 164](#)

MDFT		- Use MDFT mode.
RhsScale	(Table 133)=<float>	Scaling of RHS in Newton (MDFT mode only).
SolveSpectrum(Name=<string>)	Solve spectrum with (mandatory) name.
	{<int>*n } ...}	List of spectrum multi-indices. n is the number of tones. Solve Spectrum on page 167
UpdateScale	(Table 133)=<float>	Scaling of update in Newton (MDFT mode only).
ValueMin	(Table 133)=<float>	Lower bound for quantity in time domain (MDFT mode only).
ValueVariation	(Table 133)=<float>	Allowed variation of quantity in time domain (MDFT mode only).

Table 154 Linear solvers (refer to *Solvers User Guide*)

ILS		Parallel, iterative linear solver, for single-device or multiple-device problems with over 8000 vertices.
(MultipleRHS	- Solve linear systems with multiple right-hand sides (only for AC analysis).
	Set=<int>	Use ILS options from set <int>.
ParDiSo		Parallel, supernodal direct solver, for single-device problems with meshes up to 10 000 vertices.
	IterativeRefinement	- Perform up to two iterative refinement steps to improve the accuracy of the solution.
	IterativeRefinement=<int>	Perform up to <int> iterative refinement steps.
	MultipleRHS	- Solve linear systems with multiple right-hand sides (only for AC analysis).
	NonsymmetricPermutation	+ Compute an initial nonsymmetric matrix permutation and scaling, which places large matrix entries on the diagonal.
	RecomputeNonsymmetricPermutation)	- Compute a nonsymmetric matrix permutation and scaling before each factorization.
Slip		Iterative solver, for single-device problems with over 8000 vertices.
Super		Supernodal direct solver, for single-device problems with meshes up to 10000 vertices.
UMF		UMFPACK solver, for single-device or multiple-device problems with up to 10000 total mesh vertices.
(UMF options)	See the <i>Solvers User Guide</i> for a list of options.

Table 155 MLDAbbox({}...)
[Modified Local-Density Approximation on page 264](#)

MaxX	=<float>	μm Upper x -coordinate of the box.
MaxY	=<float>	μm Upper y -coordinate of the box.
MaxZ	=<float>	μm Upper z -coordinate of the box.
MinX	=<float>	μm Lower x -coordinate of the box.
MinY	=<float>	μm Lower y -coordinate of the box.
MinZ	=<float>	μm Lower z -coordinate of the box.

G: Command File Overview

Math

Table 156 Nonlocal() [Nonlocal Meshes on page 104](#)

Table 237		(g) Interface that is part of the reference surface.
Barrier	(Table 236...)	- (g) Regions that form the tunneling barrier.
Direction	=<vector>	(0 0 0) (cgi) If nonzero, suppress the construction of nonlocal mesh lines with a direction more than MaxAngle degrees different from <vector>.
Discretization	=<float>	1e100 cm (cgi) Maximum distance between nonlocal mesh points on a nonlocal line.
Electrode	=<string>	(g) Electrode that is part of the reference surface.
Endpoint		+ - (r) Allow nonlocal lines that end in the region. Default is Endpoint for semiconductors; otherwise, -Endpoint.
	(Table 236...)	+ - (g) Regions where nonlocal lines can or cannot end.
Length	=<float>	cm (cgi) Distance from the interface or contact up to which nonlocal mesh lines are constructed.
MaxAngle	=<float>	180 deg (cgi) Suppress construction of nonlocal mesh lines that enclose an angle of more than <float> degrees with the vector specified by Direction.
Outside		+ (cgi) Allow nonlocal mesh lines to leave the device.
Permeable		+ (r) Allow extension of nonlocal lines (as specified by the Permeation parameter) into or across the region.
	(Table 236...)	+ (g) Regions into or across which nonlocal lines can or cannot be extended.
Permeation	=<float>	0 cm (cgi) Length by which nonlocal mesh lines are extended across the interface or contact.
Refined		+ (r) Autorefine nonlocal lines inside the region.
	(Table 236...)	+ (g) Regions in which nonlocal lines are or are not autorefined.
Transparent		+ (r) Allow nonlocal lines crossing the region.
	(Table 236...)	+ (g) Regions that nonlocal lines can or cannot cross.

Table 157 Transient time-step control Math Parameters for Transient Analysis on page 94

CheckTransientError		off Error control of transient integration method.
NoCheckTransientError		on No error control of transient integration method.
TransientDigits	=<float>	3 Relative accuracy for time-step control.
TransientError	(Table 133) = <float>	Absolute error for time-step control.
TransientErrRef	(Table 133) = <float>	Error reference for time-step control.
TrStepRejectionFactor	=<float>	Factor f_{rej} . Controlling Transient Simulations on page 903

Table 158 VoronoiFaceBoxMethod= Box Method Coefficients on page 886

MaterialBoundaryTruncated	Use truncated correction for obtuse elements that have an obtuse face on boundary of materials or that are not Delaunay.
RegionBoundaryTruncated	Use truncated correction for obtuse elements that have an obtuse face on boundary of regions or that are not Delaunay.
Truncated	Use truncated correction for all obtuse elements.
-Truncated	Use truncated correction only for non-Delaunay elements.

Physics

Table 159 Physics{} Part II on page 175

Affinity	(<string>)	(r) Use PMI model <string> for electron affinity. Electron Affinity on page 975
AlphaParticle(Table 200)	(g) Generation by alpha particles. Alpha Particles on page 490
AnalyticTEP		(g) Analytic expression for thermoelectric power. Thermoelectric Power (TEP) on page 649
Aniso(Table 203)	Anisotropic properties. Chapter 21 on page 575
AreaFactor	=<float>	1 (g) Multiplier for current and heat flux densities at electrodes and thermodes. Electrode Section on page 50
BarrierLowering		off (c) Use barrier lowering for Schottky contact. Barrier Lowering at Schottky Contacts on page 209

G: Command File Overview

Physics

Table 159 Physics{} Part II on page 175

Charge((Table 205) ...)	Oxide and interface charges. Insulator Fixed Charges on page 363
ComplexRefractiveIndex()	off (g) Use complex refractive index model. Complex Refractive Index Model on page 477
CondInsulator		(r) Turn an insulator into a conductive insulator. Conductive Insulators on page 195
DefaultParametersFromFile		Initialize default parameters from parameter files instead of using built-in values. Default Parameters on page 120
DistResistance	=<float>	Ωcm^2 (i) Distributed resistance at metal–semiconductor interface. Conductivity of Metals on page 193
	=SchottkyResist	Emulate a Schottky interface. Conductivity of Metals on page 193
eBarrierTunneling	<string> [(Table 166)]	off (g) Nonlocal tunneling from and to conduction band. Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 518
EffectiveIntrinsicDensity	(Table 207)	(r) Band gap and bandgap narrowing. Band Gap and Electron Affinity on page 225
EffectiveMass	(GaussianDOS)	(r) Use Gaussian density-of-states model for organic semiconductors. Gaussian Density-of-States for Organic Semiconductors on page 240
	(<string>)	(r) Use PMI model <string> for effective mass. Effective Mass on page 978
eMLDA	[()]	- (r) MLDA model for electrons. Modified Local-Density Approximation on page 264
	LambdaTemp)	+ (r) Use temperature-dependent λ for both electrons and holes.
eMobility(Table 198)	(r) Electron mobility model. Chapter 8 on page 267
EMWGeneration(Table 168)	Sentaurus Device Electromagnetic Wave Solver (EMW) generation. Finite-Difference Time-Domain Method on page 447
eMultiValley		off (r) Multivalley statistics. Multivalley Statistics on page 203 , Multivalley Band Structure on page 605

Table 159 Physics{} Part II on page 175

EnergyRelaxationTimes	(<string>)	Use PMI model <string> to compute energy relaxation times. Energy Relaxation Times on page 981
	(constant)	Use constant energy relaxation times.
	(formula)	Use the value of formula in the parameter file.
	(irrational)	Use the ratio of two irrational polynomials. Energy-dependent Energy Relaxation Time on page 565
eQCvanDort		off (r) van Dort model for electrons. van Dort Quantization Model on page 252
eQuantumPotential	[(Table 208)]	- (r) Activate electron density gradient quantum correction. Density Gradient Quantization Model on page 260
eQuasiFermi	=<float>	V (r) Initial quasi-Fermi potential specification for electrons. Regionwise Specification of Initial Quasi-Fermi Potentials on page 220
eRecVelocity	=<[0,)>	2.573e6 cms ⁻¹ (i) Electron recombination velocity. Conductivity of Metals on page 193
eThermionic		(i) Thermionic emission model for electrons. Conductive Insulators on page 195 , Thermionic Emission Current on page 559
	(Organic_Gaussian)	(i) Thermionic-like Gaussian emission model at organic heterointerfaces for electrons. Gaussian Transport Across Organic Heterointerfaces on page 561
Fermi		off (g) Fermi statistics. Fermi Statistics on page 202
	(- WithJoyceDixon)	(g) Fermi statistics with old Wuensche approximation for Fermi integrals. Fermi Statistics on page 202
GateCurrent(Table 210)	(i) Gate currents (hot-carrier injection, some of the tunneling models).
hBarrierTunneling	<string> [(Table 166)]	off (g) Nonlocal tunneling from and to valence band. Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 518
HeatCapacity(Table 209)	(r) Heat capacity.
HeavyIon(Table 201)	off (g) Generation by heavy ions. Heavy Ions on page 492
HeteroInterface		(i) Double points at interfaces. Abrupt and Graded Heterojunctions on page 558

G: Command File Overview

Physics

Table 159 Physics{} Part II on page 175

hMLDA ([()]	- (r) MLDA model for holes. Modified Local-Density Approximation on page 264
	LambdaTemp)	+ (r) Use temperature-dependent λ for both electrons and holes.
hMobility(Table 198)	(r) Hole mobility model. Chapter 8 on page 267
hMultiValley		off (r) Multivalley statistics. Multivalley Statistics on page 203 , Multivalley Band Structure on page 605
hQCvanDort		off (r) van Dort model for holes. van Dort Quantization Model on page 252
hQuantumPotential	[(Table 208)]	- (r) Activate hole density gradient quantum correction. Density Gradient Quantization Model on page 260
hQuasiFermi	=<float>	V (r) Initial quasi-Fermi potential specification for holes. Carrier Injection with Explicitly Evaluated Boundary Conditions for Continuity Equations on page 545
hRecVelocity	=< [0 ,) >	1.93×10^6 cms ⁻¹ (i) Hole recombination velocity. Conductivity of Metals on page 193
hThermionic		(i) Thermionic emission model for holes. Thermionic Emission Current on page 559 , Conductive Insulators on page 195
	(Organic_Gaussian)	(i) Thermionic-like Gaussian emission model at organic heterointerfaces for holes. Gaussian Transport Across Organic Heterointerfaces on page 561
Hydrodynamic	[()]	(g) Hydrodynamic model for electrons and holes. Hydrodynamic Transport Model on page 184
	(eTemperature)	(g) Hydrodynamic model for electrons only. Hydrodynamic Transport Model on page 184
	(hTemperature)	(g) Hydrodynamic model for holes only. Hydrodynamic Transport Model on page 184
HydrogenDiffusion		off (ri) Hydrogen transport model without reaction specification. Using Hydrogen Transport Degradation Model on page 385
	(Table 210 ...)	off (ri) Hydrogen transport model with reaction specification. Using Hydrogen Transport Degradation Model on page 385
IncompleteIonization(Table 211)	Incomplete ionization. Chapter 6 on page 245
Laser(Table 182)	Laser. Lasers on page 687

Table 159 Physics{} Part II on page 175

LatticeTemperatureLimit	=<float>	K Maximum lattice temperature. Break Criteria on page 99
LED(Table 192)	LED. Lasers on page 687
MagneticField	=<vector>	T Magnetic field. Chapter 24 on page 643
Mobility(Table 198)	(r) Mobility model. Chapter 8 on page 267
MoleFraction(Table 214)	Mole fractions. Mole-Fraction Specification on page 550
MSConfigs(Table 215 ...)	(r) Multistate configurations. Specifying Multistate Configurations on page 367
MSPeltierHeat		(i) Peltier heat at metal–semiconductor interfaces. Thermodynamic Transport Model on page 179 , Using the Thermodynamic Model on page 180
MultiValley		off (r) Multivalley statistics. Multivalley Statistics on page 203 , Multivalley Band Structure on page 605
Noise(Table 216)	(r) Noise sources. Noise Sources on page 501
OptBeam((Table 172) ...)	Photon beam generation. Optical Beam Absorption on page 434
OpticalGeneration{	Table 173}	Optical generation by TMM. Transfer Matrix Method on page 438
OpticalGenerationFromFile(Table 175)	Modify the optical generation profile loaded from a file.
Piezo(Table 219)	Piezoresistive model. Stress-induced Electron Mobility Model on page 617
Piezoelectric_Polarization	(<string>)	Use PMI model <string> to compute piezoelectric polarization. Piezoelectric Polarization on page 1013
	(strain)	Use strain model to compute piezoelectric polarization. Strain Model on page 636
	(stress)	Use stress model to compute piezoelectric polarization. Stress Model on page 636
Polarization		off (r) Use ferroelectric model. Chapter 22 on page 589
	(Memory=<2...>)	10 Maximum nesting of minor loops. Using Ferroelectrics on page 589

G: Command File Overview

Physics

Table 159 Physics{} Part II on page 175

PostTemperature		(g) Use simplified self-heating model. Uniform Self-Heating on page 182 , Using the Thermodynamic Model on page 180
	*	Compute dissipated power as integral of Joule heat over entire device.
	(IV_diss)	(g) Compute dissipated power as $\sum IV$ over all electrodes.
	(IV_diss(<string>...))	(g) Compute dissipated power as $\sum IV$ over user-selected electrodes.
Radiation()	Table 202	Radiation model. Chapter 15 on page 489
RayTrace((Table 176) ...)	Raytracing. Raytracing on page 407
RecGenHeat((g) Generation–recombination processes act as heat sources. Hydrodynamic Transport Model on page 184
(OptGenOffset=<float>	0 . 5 Divide contribution of optical generation rate into energy gain/loss terms H_n and H_p . Hydrodynamic Model on page 184
	OptGenWavelength=<float>	μm Wavelength if optical generation is loaded from file. Hydrodynamic Model on page 184
Recombination()	Table 160)	Generation–recombination model. Chapter 9 on page 309
Schottky		off (i) Use Schottky boundary conditions. Conductivity of Metals on page 193
Schroedinger	(Table 221)	(i) Schrödinger solver. 1D Schrödinger Solver on page 253
	<string> (Table 221)	(g) Schrödinger solver on named nonlocal mesh.
SingletExciton(off (ri) Activate region or interface for singlet exciton equation.
(FluxBC	off (i) Impose a zero flux boundary condition on specified interface. Using the Singlet Exciton Equation on page 191
	BarrierType(Table 204)	(i) Set the barrier type at organic heterointerface. Using the Singlet Exciton Equation on page 191
	Recombination(Table 177))	off (r) Switch on generation and recombination terms in singlet exciton equation. Using the Singlet Exciton Equation on page 191

Table 159 Physics{} Part II on page 175

TaileDistribution		off (r) Specify the region where the tail electron distribution is calculated. Using Tail Distribution Hot-Carrier Injection Model on page 541
TailhDistribution		off (r) Specify the region where the tail hole distribution is calculated. Using Tail Distribution Hot-Carrier Injection Model on page 541
Temperature	=<float>	300 K (g) Device (lattice) temperature.
ThermalConductivity(Table 222)	(r) Thermal conductivity.
Thermodynamic		off (g) Thermodynamic transport model. Uniform Self-Heating on page 182
Traps ((Table 224) ...)	Traps. Chapter 10 on page 349
UseNitrogenAsDopant		off (g) Recognize nitrogen as a dopant. Material and Doping Specification on page 123

Generation and Recombination

Table 160 Recombination() [Chapter 9 on page 309](#)

<string>		Use PMI model <string>. Generation–Recombination Model on page 933
Auger		off (r) Auger recombination. Auger Recombination on page 322
	(WithGeneration)	off (r) Auger recombination and generation.
Avalanche(Table 162)	off (r) Impact ionization. Avalanche Generation on page 324
Band2Band(Table 163)	off (r) Band-to-Band Tunneling Models on page 335
CDL(Table 178)	off (r) Coupled defect level recombination. Coupled Defect Level (CDL) Recombination on page 320
eAvalanche(Table 162)	off (r) Electron impact ionization. Avalanche Generation on page 324
eBarrierTunneling	[(Table 166)]	off (ci) Nonlocal tunneling from and to conduction band. Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 518
hAvalanche(Table 162)	off (r) Hole impact ionization. Avalanche Generation on page 324

G: Command File Overview

Physics

Table 160 Recombination() [Chapter 9 on page 309](#)

hBarrierTunneling	[Table 166]	off (ci) Nonlocal tunneling from and to valence band. Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 518
Radiative		- (r) Radiative recombination. Radiative Recombination on page 321
SRH (Table 178)	off (r) Shockley–Read–Hall recombination. Shockley–Read–Hall Recombination on page 309
SurfaceSRH		off (i) Interface Shockley–Read–Hall recombination. Surface SRH Recombination on page 319
TrapAssistedAuger		off (r) Trap-assisted Auger recombination. Trap-assisted Auger Recombination on page 318

Table 161 AutoMatGen() [Finite-Difference Time-Domain Method on page 447](#), Material/
[AutoMatGen on page 453](#)

RefractiveIndex	(model=<string>)	Name of PMI model for refractive index. Refractive Index on page 1000
	(model=ODB)	Use parameters from the TableODB section in the parameter file. Table-based Optical Properties of Materials in Parameter File on page 475
	(model=Parameter)	Compute refractive index according to Refractive Index Models on page 472 .
	(value=<float>)	Constant value n for the refractive index.
SemAbs	(model=<string>)	Name of PMI model for absorption coefficient. Optical Absorption on page 998
	(model=ODB)	Use parameters from the TableODB section in the parameter file. Table-based Optical Properties of Materials in Parameter File on page 475
	(model=Parameter)	* Compute absorption coefficient according to Default Absorption Model from Parameter File on page 473 .
	(model=RSS)	Use the silicon absorption model. Absorption Coefficient Model on page 475
	(value=<float>)	cm^{-1} Constant value α for the absorption coefficient.

Table 162 Avalanche() Avalanche Generation on page 324

<string>	Use PMI model <string>. Avalanche Generation Model on page 935
CarrierTempDrive	Use temperature driving force. Avalanche Generation with Hydrodynamic Transport on page 331
ElectricField	Use plain electric field driving force. Approximate Breakdown Analysis: Poisson Equation Approach on page 333
Eparallel	Use current-parallel electric field driving force. Driving Force on page 331
GradQuasiFermi	* Use gradient quasi-Fermi potential driving force. Driving Force on page 331
Lackner	Use Lackner model. Lackner Model on page 326
Okuto	Use Okuto–Crowell model. Okuto–Crowell Model on page 326
UniBo	Use University of Bologna model. University of Bologna Impact Ionization Model on page 327
UniBo2	Use new University of Bologna impact ionization model. New University of Bologna Impact Ionization Model on page 329
vanOverstraeten	* Use van Overstraeten model. van Overstraeten – de Man Model on page 325

Table 163 Band2Band() Band-to-Band Tunneling Models on page 335

DensityCorrection	=Local	Use density correction. Schenk Density Correction on page 337
	=None	* Use plain densities. Schenk Density Correction on page 337
FranzDispersion		- Use Franz dispersion in the direct nonlocal path model. Using Nonlocal Path Band-to-Band Model on page 342
InterfaceReflection		+ Consider interface reflection in the nonlocal path model. Using Band-to-Band Tunneling on page 335

G: Command File Overview

Physics

Table 163 Band2Band() Band-to-Band Tunneling Models on page 335

Model	=E1	Use simple model with $P = 1$. Simple Band-to-Band Models on page 337
	=E1_5	Use simple model with $P = 1.5$. Simple Band-to-Band Models on page 337
	=E2	Use simple model with $P = 2$. Simple Band-to-Band Models on page 337
	=Hurkx	Use Hurkx model. Hurkx Band-to-Band Model on page 338
	=NonlocalPath1	Use nonlocal path model with first tunnel path. Dynamic Nonlocal Path Band-to-Band Model on page 339
	=NonlocalPath2	Use nonlocal path model with first and second tunnel paths. Dynamic Nonlocal Path Band-to-Band Model on page 339
	=NonlocalPath3	Use nonlocal path model with first, second, and third tunnel paths. Dynamic Nonlocal Path Band-to-Band Model on page 339
	=Schenk	Use Schenk model. Schenk Model on page 336

Table 164 BPMScalar() Beam Propagation Method on page 462

Bidirectional(Error=<float>	! Relative error used as a break criterion for iterative algorithm in bidirectional beam propagation method.
	Iterations=<int>	! Maximum number of iterations used as a break criterion for iterative algorithm in bidirectional beam propagation method.
Boundary(GridNodes=<float>*2	! Number of PML boundary grid nodes to be inserted at left and right sides.
	Order=<1..2>	! Order of spatial variation of complex stretching parameter.
	Side=<string>	! "X", "Y", or "Z".
	StretchingParameterImag=<float>*2)	! Minimum and maximum values of imaginary part of stretching parameter.
	StretchingParameterReal=<float>*2)	! Minimum and maximum values of real part of stretching parameter.
	Type="PML"	! Use PML boundary conditions.
	VacuumGridNodes=<float>*2)	! Number of vacuum grid nodes to be inserted at left and right sides.

Table 164 `BPMScalar()` Beam Propagation Method on page 462

Excitation(Amplitude=<float>	! V/m Amplitude of optical field.
	CenterGauss=<vector>	μm Gaussian center. Size of vector is $d - 1$.
	Phi=<float>	degree Angle between positive y-axis and projection of propagation vector on xy plane in 3D.
	SigmaGauss=<vector>	μm Gaussian half width. Size of vector is $d - 1$.
	Theta=<float>	degree Angle between positive y-axis (z-axis) and propagation vector in 2D (3D).
	TruncationPositionX=<float>*2)	Left and right truncation positions of plane wave for x-axis.
	TruncationPositionY=<float>*2)	Left and right truncation positions of plane wave for y-axis.
	TruncationSlope=<vector>	Plane-wave truncation slope. Size of vector is $d - 1$.
	Type="Gaussian"	Use Gaussian excitation.
	Type="PlaneWave"	Use truncated plane wave excitation.
	WaveIntensity=<float>	! $\text{m}^{-2} \text{s}^{-1}$ Wave intensity.
	Wavelength=<float>	! μm Wavelength.
	WavePower=<float>)	! W m^{-2} Wave power.
GridNodes	=<vector>	! Number of grid nodes in each spatial dimension.
ReferenceRefractiveIndex	=<float>	! Value for reference refractive index. General on page 464
ReferenceRefractiveIndexDelta	=average	Use average refractive index in propagation plane as reference refractive index.
	=fieldweighted	Use field-weighted refractive index in propagation plane as reference refractive index.
	=maximum	Use maximum refractive index in propagation plane as reference refractive index.
	=<float>	<code>ReferenceRefractiveIndexDelta</code> is added to <code>ReferenceRefractiveIndex</code> in the calculation. To be used for fine-tuning the numerics. General on page 464

G: Command File Overview

Physics

Table 165 CircularWindow{} and RectangularWindow{} [Raytracing on page 407](#), [Window of Starting Rays on page 414](#)

CellSize	=<float>	µm Spacing between rays. The definition of CellSize varies, depending on the dimension of the simulation and the window type.
intposition1	=[<float>...]	Positions of the values specified in intvalue1 function. Spatial Distribution of Intensity on page 418
intposition2	=[<float>...]	Positions of the values specified in intvalue2 function. Spatial Distribution of Intensity on page 418
intpositionnormalized1	=[<float>...]	Normalized positions of the values specified in intvalue1 function. Spatial Distribution of Intensity on page 418
intpositionnormalized2	=[<float>...]	Normalized positions of the values specified in intvalue2 function. Spatial Distribution of Intensity on page 418
intvalue1	=[<float>...]	Value for the spatially varying intensity spline function. Spatial Distribution of Intensity on page 418
intvalue2	=[<float>...]	Value for the spatially varying intensity spline function. Spatial Distribution of Intensity on page 418
NumberOfRays	=<int>	Number of rays entering the window of a 2D device. Two-dimensional Device and Window Description on page 415
WindowCenter	=<vector>	µm Position of the center of the window.
WindowDirection	=<vector>	Direction that the window faces.
WindowHeight	=<float>	µm Rectangular window height for 3D structure. Three-dimensional Device and Window Description on page 416
WindowOrientation	=<vector>	µm Alignment of the lattice in 3D. Three-dimensional Device and Window Description on page 416
WindowRadius	=<float>	µm Circular window radius for structures of any dimension.
WindowSize	=<float>	µm Window size for 2D structure. Two-dimensional Device and Window Description on page 415
WindowWidth	=<float>	µm Rectangular window width for 3D structure. Two-dimensional Device and Window Description on page 415

Table 166 eBarrierTunneling() and hBarrierTunneling() [Nonlocal Tunneling at Interfaces, Contacts, and Junctions on page 518](#)

Band2Band	=None	* No band-to-band tunneling.
	=Full	Include band-to-band tunneling with consistent parallel momentum integral with the direct nonlocal path band-to-band tunneling model.
	=Simple	Include band-to-band tunneling with simple parallel momentum integral.
	=UpsideDown	Include band-to-band tunneling with unphysical parallel momentum integral.
BandGap		- Allow tunneling into the band gap at the interface for which tunneling is specified. Use this option for backward compatibility only.
PeltierHeat		- Include Peltier heating terms for tunneling carriers. Eq. 529, p. 529
Schroedinger		- Schrödinger equation-based model instead of WKB for tunneling probabilities. This option does not work with the TwoBand or Band2Band option. Schrödinger Equation-based Tunneling Probability on page 526
Transmission		- Use additional interface transmission coefficients. Eq. 520, p. 525
TwoBand		- Use two-band dispersion relation. Eq. 519, p. 524

Table 167 ElectricField() in SRH() and CDL() [SRH Field Enhancement on page 313](#)

DensityCorrection	=Local	(r) Use density correction. Schenk TAT Density Correction on page 316
	=None	* (r) Use local densities.
Lifetime	=Constant	* (r) No field-enhanced lifetime.
	=Hurkx	(r) Use Hurkx model for lifetime enhancement. Hurkx TAT Model on page 316
	=Schenk	(r) Use Schenk model for lifetime enhancement. Schenk Trap-assisted Tunneling (TAT) Model on page 314

G: Command File Overview

Physics

Table 168 EMWGeneration() Finite-Difference Time-Domain Method on page 447

AccelewareOption		Activate EMW interface to the Acceleware hardware-accelerated FDTD kernel.
AutoMatGen(Table 161)	
Boundary(Order=<1..4>	2 Order of boundary conditions (for Mur, only up to 2). Boundary on page 450
	Side="<side>"	a11 Side to which boundary condition is applied. Valid specifications are a11, X, Xmin, Xmax, Y, Ymin, Ymax, Z, Zmin, and Zmax. Boundary on page 450
	Type="<type>"	Higdon boundary condition type. <type> can be Higdon, Mur, Periodic, PeriodicMur, or PML. Boundary on page 450
Detector(Tolerance=<float>)	Specify a tolerance for the termination criterion.
Excitation(Amplitude=<float>	Vm ⁻¹ Excitation on page 451
	Phi=<float>	degree Angle between positive x-axis and the plane wave propagation direction in 3D. Excitation on page 451
	Psi=<float>	degree Polarization angle. Excitation on page 451
	Theta=<float>	degree Angle between positive y-axis (z-axis) and the plane wave propagation direction in 2D (3D). Excitation on page 451
	WaveLength=<float>	m Excitation on page 451
	WavePower=<float>)	Wm ⁻² Excitation on page 451
Globals(NumberOfThreads=<int>	Number of threads used for multithreading.
	TimeStep=<float>	s Time step used in FDTD algorithm.
	TotalTimeSteps=<int>	
	TotalSimulationTime=<float>)	s
GridBoundX	=(<float>*2)	GridBoundX, GridBoundY, and GridBoundZ on page 457
GridBoundY	=(<float>*2)	GridBoundX, GridBoundY, and GridBoundZ on page 457
GridBoundZ	=(<float>*2)	GridBoundX, GridBoundY, and GridBoundZ on page 457

Table 168 EMWGeneration() [Finite-Difference Time-Domain Method on page 447](#)

Material(Conductivity=<float>	$0 \Omega^{-1} m^{-1}$
	Density=<float>	$0 m^{-3}$
	MagneticConductivity=<float>	0
	Name=<string>	
	Permeability=<float>	1 Relative permeability.
	Permittivity=<float>	1 Relative permittivity.
	SpecificHeat=<float>	0
	ThermalConductivity=<float>	$0 W m^{-1} K^{-1}$
NodePerWavelength	=<int>	10 Minimum number of nodes that is placed in a wavelength in all directions.
NodePerWavelengthX	=<int>	Similar to NodePerWavelength but only for the <i>x</i> direction.
NodePerWavelengthY	=<int>	Similar to NodePerWavelength but only for the <i>y</i> direction.
NodePerWavelengthZ	=<int>	Similar to NodePerWavelength but only for the <i>z</i> direction.
Plot(Tickstep=<int>)	A plot is made at every TickStep-th time step.
SmoothingFactor	=<float>	SmoothingFactor on page 459

Table 169 FDTD() [Specifying the Optical Solver on page 400](#)

AccelewareOption		off Activate EMW interface to the Acceleware hardware-accelerated FDTD kernel.
GenerateMesh(Control of tensor mesh generation using Sentaurus Mesh.
	Once)	Generate tensor mesh once at the beginning of the simulation.
	ForEachWavelength)	Generate tensor mesh whenever the wavelength changes compared with the previous FDTD solution.
	Wavelength=(<float>*<0..m>)	Specify list of strictly monotonically increasing wavelengths between which the tensor mesh is generated only once.

G: Command File Overview

Physics

Table 170 Layers()... Transfer Matrix Method on page 438 (deprecated transfer matrix method)

Absorption	=<float>	Constant value of absorption coefficient (overwrites the value from the TableODB section in the parameter file).
BottomReflectivity	=<[0,1]>	Bottom reflectivity of the current layer (overwrites the value from the TableODB section in the parameter file).
Left	=<vector>	'Upper-left' corner of the current layer illuminated by light wave. Needed for 1D simulation.
Middle	=<vector>	'Upper-middle' corner of current layer illuminated by light wave. Needed for 3D simulation along with Left and Right vectors.
Refract	=<float>	Constant value of refractive index (overwrites the value from the TableODB section in the parameter file).
Right	=<vector>	'Upper-right' corner of the current layer illuminated by light wave. Needed for 2D simulation along with Left vector.
SiLayer		Calculate and store generation rates for this layer. Must be used in one layer only.
Thickness	=<float>	µm Thickness of current layer.
TopReflectivity	=<[0,1]>	Top reflectivity of the current layer (overwrites the value from the TableODB section in the parameter file).

Table 171 Medium() Transfer Matrix Method on page 438

ExtinctionCoefficient	=<float>	1
Location	=top	! Position of medium with respect to extracted layer structure.
	=bottom	Position of medium with respect to extracted layer structure.
Material	=<string>	Name of material.
RefractiveIndex	=<float>	1

Table 172 OptBeam(...) Optical Beam Absorption on page 434

SemAbs	(material=<string>)	Use TableODB for material <string> for all regions where the beam is defined. Table-based Optical Properties of Materials in Parameter File on page 475
	(model=<string>)	Name of PMI model for absorption coefficient. Optical Absorption on page 998
	(model=ODB)	Use parameters from the TableODB section in the parameter file. Table-based Optical Properties of Materials in Parameter File on page 475
	(model=Parameter)	* Compute absorption coefficient according to default models in parameter file. Default Absorption Model from Parameter File on page 473
	(model=RSS)	Use the silicon absorption model. Absorption Coefficient Model on page 475
	(value=<float>)	cm ⁻¹ Constant value α for the absorption coefficient. Wave properties of the beam are not required. Eq. 432, p. 436
SemSurf	=<float>	cm Coordinate z_0 of the semiconductor surface. Eq. 432, p. 436
SemVelocity	=(<float>*2)	cm s ⁻¹ Velocity of window perpendicular to the direction of the incident beam.
SemWindow	=(<float>*2)(<float>*2)	cm Larger (x_{\max}, y_{\max}) and smaller (x_{\min}, y_{\min}) coordinates of semiconductor window for 3D.
	=(<float>*2)	cm Larger and smaller coordinate (x_{\min}, x_{\max}) of semiconductor window for 2D.
WaveEnergy	=<float>	eV Photon energy. Eq. 430, p. 435
WaveInt	=<float>	cm ⁻² s ⁻¹ Incident beam intensity. Eq. 430, p. 435
WaveLength	=<float>	cm Photon wavelength. Eq. 430, p. 435
WavePower	=<float>	W cm ⁻² Incident wave power. Eq. 430, p. 435
WaveTime	=(<float>*2)	s Time interval (t_{\min}, t_{\max}) when the incident beam intensity is constant. Eq. 432, p. 436
WaveTsigma	=<float>	s Standard deviation σ_t of the temporal Gaussian distribution that describes the decay of the incident beam intensity outside the time interval WaveTime. Eq. 432, p. 436
WaveXYsigma	=<float>	cm Standard deviation σ_{xy} of the spatial Gaussian distribution that describes the decay of the incident beam intensity outside the defined semiconductor window.

G: Command File Overview

Physics

Table 173 OpticalGeneration{} Transfer Matrix Method on page 438 and Beam Propagation Method on page 462

Area	=<float>	
ComputeFromSpectrum		Compute optical generation for a given illumination spectrum. Spectral Illumination on page 486
Lambda	=<float>	μm Characteristic spreading length λ_{sp} . Eq. 444, p. 442
Layers ((Table 170) ...	Layer entry.
	FacetAngle=<float>	degree
	InitialLayerRefract=<float>)	Refractive index of the layer in which the light wave starts.
Light (Direction=<vector>	
	Intensity=[<float>...]	W cm^{-2} Intensity list (matches wavelength list).
	Polarization=<[0,1]>	
	Wavelength=[<float>...])	μm Wavelength list (matches intensity list).
Loss	=<float>	Loss factor β . Eq. 443, p. 441
MultipleReflections	=include	
	=none	
QuantumYield	=<float>	Eq. 439, p. 441
Shift	=<vector>	
StandingWaves	=include	
	=none	

Table 174 OpticalGeneration{} Specifying the Type of Optical Generation Computation on page 394

AutomaticUpdate		+ Controls whether optical generation is recomputed if quantities on which it depends are not up-to-date.
ComputeFromMonochromaticSource(Optical Generation from Monochromatic Source on page 395
	TimeDependence(Table 179)	
	Scaling=<float>)	1
ComputeFromSpectrum(Illumination Spectrum on page 396
	TimeDependence(Table 179)	
	Scaling=<float>)	1
ReadFromFile(Loading and Saving Optical Generation from and to File on page 397
	TimeDependence(Table 179)	
	Scaling=<float>)	1
Scaling	=<float>	1
SetConstant(Constant Optical Generation on page 397
	TimeDependence(Table 179)	
	Value=<float>)	s ⁻¹ cm ⁻³ Value for optical generation rate.
TimeDependence(Table 179)	Specification of type of time dependency.

Table 175 OpticalGenerationFromFile() Loading Optical Generation from File on page 483

OptScaling	=<float>	1 Scaling for the loaded optical generation profile.
WaveTime	=(<float>*2)	Start-time and end-time specification of the constant interval.
WaveTSigma	=<float>	The variance of the time Gaussian pulse used.

G: Command File Overview

Physics

Table 176 RayTrace((...)) in Physics{} [Raytracing on page 407](#)

CircularWindow{	Table 165	Properties of the window that allows rays through. Window of Starting Rays on page 414
DepthLimit	=<int>	Stop tracing a ray after passing through more than <int> material boundaries.
MinIntensity	=<float>	Stop tracing a ray when its intensity becomes less than <float> times the original intensity.
MonteCarlo		Activate Monte Carlo raytracing.
OmitReflectedRays		Discard all reflected rays from raytracing process.
OmitWeakerRays		Discard the weaker ray at a material interface. This is chosen by comparing the reflectivity and transmittivity.
OptGenScaling	= <float>	Multiplication factor to the optical generation profile computed in raytracing process.
Polarization(Axis1=<float>	Relative length of one axis. Polarization of Old Raytracer on page 434
	Axis1Dir=<vector>	Direction of Axis1. Axis2 is perpendicular to Axis1 and direction of the ray. Polarization of Old Raytracer on page 434
	Axis2=<float>)	Relative length of the other axis. Polarization of Old Raytracer on page 434
PolarizationVector	= Random	Default. Automatically choose a random polarization vector that is perpendicular to the starting ray direction.
	= <vector>	Set a fixed polarization vector for the starting ray.
Print		Create a .grd file with information about the raytree
RectangularWindow{	Table 165	Properties of the window that allows rays through. Window of Starting Rays on page 414
RedistributeStoppedRays		Distribute the total power accumulated at terminated rays back into the raytree.
RefractiveIndex	(model=<string>)	Name of PMI model for refractive index.
	(model=ODB)	Use parameters from the TableODB section in the parameter file. Table-based Optical Properties of Materials in Parameter File on page 475
	(model=Parameter)	Compute refractive index according to Refractive Index Models on page 472 .
	(value=<float>)	Constant value for the refractive index n .

Table 176 RayTrace(...) in Physics{} [Raytracing on page 407](#)

SemAbs	(material=<string>)	Use TableODB for material <string> for all regions where the ray is defined. Table-based Optical Properties of Materials in Parameter File on page 475
	(model=<string>)	Name of PMI model for absorption coefficient. Optical Absorption on page 998
	(model=ODB)	Use parameters from the TableODB section in the parameter file. Table-based Optical Properties of Materials in Parameter File on page 475
	(model=Parameter)	* Compute absorption coefficient according to default models in parameter file. Default Absorption Model from Parameter File on page 473
	(model=RSS)	Use the silicon absorption model. Absorption Coefficient Model on page 475
	(value=<float>)	cm ⁻¹ Constant value α for the absorption coefficient.
TurnOffTreeNodeCount		Disable counting the total number of tree nodes.
WaveDirection	=<vector>	Direction of the rays.
WaveEnergy	=<float>	eV Photon energy.
WaveInt	=<float>	cm ⁻² s ⁻¹ Ray intensity.
WaveLength	=<float>	cm Ray wavelength.
WavePower	=<float>	Wcm ⁻² Ray power.
WaveTime	=(<float>*2)	s Time interval (t_{\min}, t_{\max}) when the ray intensity is constant.
WaveTsigma	=<float>	s Standard deviation σ_t of the temporal Gaussian distribution that describes the decay of the ray intensity outside the time interval WaveTime.

G: Command File Overview

Physics

Table 177 Recombination() in SingletExciton()

Bimolecular		off (r) Switch on bimolecular recombination in continuity and singlet exciton equations. Singlet Exciton Equation on page 189 , Bimolecular Recombination on page 344
eQuenching		off (r) Switch on quenching of singlet exciton due to free electrons. Singlet Exciton Equation on page 189
hQuenching		off (r) Switch on quenching of singlet exciton due to free holes. Singlet Exciton Equation on page 189
radiative		off (r) Switch on directly radiative decay of singlet exciton associated with light emission.
trappedradiative		off (r) Switch on trap-assisted radiative decay of singlet exciton associated with light emission.

Table 178 SRH() and CDL() Shockley–Read–Hall Recombination on page 309, Coupled Defect Level (CDL) Recombination on page 320

<string>		Use PMI model <string> to compute lifetimes. Lifetimes on page 985
ElectricField(Table 167)	(r) Field enhancement. SRH Field Enhancement on page 313
DopingDependence		Doping dependence. SRH Doping Dependence on page 311
ExpTempDependence		Exponential temperature dependence. SRH Temperature Dependence on page 312
TempDependence		Temperature dependence. SRH Temperature Dependence on page 312

Table 179 TimeDependence() Specifying Time Dependency for Transient Simulations on page 398

FromFile		off Reads the time dependency as a table from file.
WaveTime	=(<float>*2)	s Time interval (t_{\min} , t_{\max}) when the optical generation rate is constant.
WaveTSigma	=<float>	s Standard deviation σ_t of the temporal Gaussian distribution that describes the decay of the optical generation rate outside the time interval WaveTime.
WaveTSlope	=<float>	s^{-1} Slope that characterizes the linear decay of the optical generation rate outside the time interval WaveTime.

Table 180 TMM() Transfer Matrix Method on page 438

Excitation(Amplitude=<float>	! V/m Amplitude of optical field.
	Polarization=<float>	Polarization of incident light.
	Theta=<float>	degree Angle between positive y-axis and propagation vector in 2D.
	WaveIntensity=<float>	! m ⁻² s ⁻¹ Wave intensity.
	Wavelength=<float>	! μm Wavelength.
	WavePower=<float>)	! Wm ⁻² Wave power.
IntensityPattern		Choose type of intensity pattern.
	=StandingWave	* Compute the regular optical intensity without applying any algorithm that filters out oscillations on the wavelength scale.
	=Envelope	Compute the envelope of the optical intensity instead of the regular optical intensity.
NodesPerWavelength	=<float>	Number of nodes per wavelength used for computation of optical field.
Stripe(Left=<float>	! μm Left position of stripe.
	Right=<float>	! μm Right position of stripe.
	Medium(Table 171)	Specification of media surrounding the extracted layer structure.

Laser and LED

Table 181 Laser() or LED() Chapter 28 on page 721

BandStructure(Table 184)	k · p band structure model. The k.p Method on page 825
Broadening(Table 185)	Activate gain-broadening models or nonlinear gain saturation model. Gain-broadening Models on page 814 , Nonlinear Gain Saturation Effects on page 815
EmissionTable()	Specify the parameterization of dephasing rate and tabulated emission computation. Computing Emission Tables on page 844
eQWMobility	=<float>	cm ² V ⁻¹ s ⁻¹ Mobility for bound electrons for scattering model. Carrier Capture in Quantum Wells on page 804
FreeCarr		Use free carrier absorption. Free Carrier Loss on page 702
GroupRefract	=<float>	Fixed effective index; ignore effective index computed by the optical solvers. Edge-emitting Lasers on page 703
hQWMobility	=<float>	cm ² V ⁻¹ s ⁻¹ Mobility for bound holes for scattering model. Carrier Capture in Quantum Wells on page 804
LateralLeakage		Activate nonperpendicular current flux of free carriers across quantum wells.
ManyBodyEffects	(Type=Born)	Second Born approximation (for tabulated emission computation only). Computing Tabulated Optical Emission Data on page 844
	(Type=FCT)	* Free carrier theory. The k.p Method on page 825
	(Type=SHF)	Screened Hartree–Fock. The k.p Method on page 825
Optics(Table 192)	Optics. Chapter 29 on page 761
QWDeep		Use scattering model for deep quantum wells. Carrier Capture in Quantum Wells on page 804
QWeScatTime	=<float>	s Electron scattering time into quantum well.
QWExtension	=AutoDetect	Autodetect width of quantum wells. The quantum-well region must be specified by the keyword Active. Carrier Capture in Quantum Wells on page 804
QWhScatTime	=<float>	s Hole scattering time into quantum well.
QWScatModel		Use scattering model for quantum well. Carrier Capture in Quantum Wells on page 804

Table 181 Laser() or LED() [Chapter 28 on page 721](#)

QWShallow		Use scattering model for flat quantum wells. Carrier Capture in Quantum Wells on page 804
QWTransport		Use ‘three-point’ QW model with thermionic emission. Carrier Capture in Quantum Wells on page 804
ReducedOutcoupling(FreeCarrierAbsorption =<float>	Parameter for phenomenological reduction of VCSEL output power due to free carrier absorption. VCSEL Output Power on page 714
	OpticalLoss=<float>)	Parameter for phenomenological reduction of VCSEL output power due to distributed background losses. VCSEL Output Power on page 714
RefractiveIndex(CarrierDep	Use carrier density-dependent refractive index. Carrier-Density Dependence of Refractive Index on page 700
	TempDep)	Use temperature-dependent refractive index. Temperature Dependence of Refractive Index on page 699
SplitOff	=<float>	eV Spin-orbit split-off energy. Strain Effects on page 820
SponEmissionCoeff	(<string>)	Use PMI model <string> for spontaneous emission. Importing Gain and Spontaneous Emission Data with PMI on page 849
	(AnalyticSponEmission Model)	Activate use of simple model for spontaneous emission. Stimulated and Spontaneous Emission Coefficients on page 809
	(Tabulated)	Activate loading of tabulated spontaneous emission data. Loading Tabulated Stimulated and Spontaneous Emission on page 848
SponIntegration	(<float>,<int>)	eV Energy integration range and number of discretization intervals for numeric integration (for LED simulations only). Spontaneous Emission Power for LEDs on page 812
SponScaling	=<float>	Scaling factor for matrix element of spontaneous emission. Radiative Recombination and Gain Coefficients on page 809
StimEmissionCoeff	(<string>)	Use PMI model <string> for stimulated emission. Importing Gain and Spontaneous Emission Data with PMI on page 849
	(Tabulated)	Activate loading of tabulated stimulated emission data. Loading Tabulated Stimulated and Spontaneous Emission on page 848

G: Command File Overview

Physics

Table 181 Laser() or LED() [Chapter 28 on page 721](#)

StimScaling	=<float>	Scaling factor for matrix element of stimulated emission. Radiative Recombination and Gain Coefficients on page 809
Strain	(RefLatticeConst=<float>)	m Use strain model for quantum well with given reference lattice constant. Strain parameters are input in the parameter file. Strain Effects on page 820

Table 182 Laser() [Chapter 27 on page 687](#)

Table 192		Keywords for lasers and LEDs.
CavityLength	=<float>	μm Cavity length. Photon Rate and Photon Phase Equations on page 690
DFB(DFBPeriod=<float>)	μm Use DFB feature with DFB grating period <float>. Edge-emitting Lasers on page 703
GainShift	=<float>	eV Shift emission energy in a laser or an LED simulation. Fitting Stimulated and Spontaneous Emission Spectra on page 813
lFacetReflectivity	=<float>	Left-facet power reflectivity. Photon Rate and Photon Phase Equations on page 690
LinewidthEnhancementFactor	=<float>	Line width enhancement factor.
LongitudinalModes		Calculate multiple longitudinal optical modes if the number of modes is specified greater than one for the applied optical solver. By default, multiple transverse modes are calculated. Edge-emitting Lasers on page 703
OpticalLoss	=<float>	cm ⁻¹ Background optical loss.
PhotonPhaseAlpha		Use line width enhancement model. This requires the specification of LinewidthEnhancementFactor.
rFacetReflectivity	=<float>	Right-facet power reflectivity.
SponEmiss	=<float>	Spontaneous emission factor.
TransverseModes		Calculate multiple transverse optical modes if the number of modes is specified greater than one for the applied optical solver. This is the default. Edge-emitting Lasers on page 703
VCSEL()	VCSEL simulation. Vertical-Cavity Surface-emitting Lasers on page 711
WaveguideLoss		off Activate the feedback of waveguide loss from optical solver to the photon rate equation.

Table 183 ActivateSpectralConversion() LED(Optics(RayTrace())) [Spectral Conversion on page 749](#)

HenyeyGreensteinScattering(<float>	Asymmetry factor of the Henyey–Greenstein scattering probability.
NumberOfPoints	=<int>	Number of energy points to represent the absorption and emission spectra of the luminescent material.
ScatterFactor	=<[0,1]>	Probability of scattering.

Table 184 BandStructure() in Laser() [The k.p Method on page 825](#)

CrystalType	=Wurtzite	The k.p Method on page 825
	=Zincblende	* The k.p Method on page 825
InternalNumkValues	=<int>	81 Internal discretization for second Born approximation. Allowed values: 81, 121, and 161. Computing Emission Tables on page 844
NumKValues	=<int>	Number of values in k space; default 16 for Zincblende, 21 for Wurtzite, and 41 for second Born approximation. Band structure is calculated up to $8 \times 10^8 \text{ m}^{-1}$ for Zincblende and up to $12 \times 10^8 \text{ m}^{-1}$ for Wurtzite.
Order	=kp4x4	The k.p Method on page 825
	=kp6x6	The k.p Method on page 825
	=kp8x8	The k.p Method on page 825 (for zinc blende only)
	=nokp	* The k.p Method on page 825
Screening		Switch on screening effect. Screening Piezoelectric Fields in GaN-type Quantum Wells on page 839

Table 185 Broadening() [Gain-broadening Models on page 814](#)

Gamma	=<float>	eV Broadening coefficient Γ .
Type	=CosHyper	Use hyperbolic-cosine broadening. Hyperbolic-Cosine Broadening on page 815
	=Landsberg	Use Landsberg broadening. Landsberg Broadening on page 814
	=Lorentzian	Use Lorentzian broadening. Lorentzian Broadening on page 814

G: Command File Overview

Physics

Table 186 DeterministicProblem() [Automatic Optical Mode Searching on page 797](#)

AzimuthalExpansion	=<int>	Order of the cylindrical harmonics, $\cos(m\phi)$ (for cavity only).
Boundary	=Type1	(For waveguide only)
	=Type2	(For waveguide only)
Coordinates	=Cartesian	(For waveguide only)
	=Cylindrical	(For cavity only)
EquationType	=Cavity	Solve a cavity problem for the resonant frequency.
	=Waveguide	Solve a frequency-domain Helmholtz equation for the effective index.
LasingWavelength	=<float>	! nm Lasing wavelength (for waveguide only).
SourcePolarization	=phiDirection	(For cavity only)
	=rhoDirection	(For cavity only)
	=xDirection	(For waveguide only)
	=yDirection	(For waveguide only)
	=zDirection	
SourcePosition	(<float>*2)	Coordinates of the source dipole.
Sweep	(<float>*2 <int>)	For cavity, start and end wavelength, in nm , and number of points; for waveguide, start and end effective index, and number of points.
Symmetry	=NonSymmetric	(For waveguide only)
	=Periodic	(For waveguide only)
	=Symmetric	(For waveguide only)

Table 187 EffectiveIndex() [Effective Index Method for VCSELs on page 777](#)

Absorption	(ODB)	Use absorption from the TableODB section of the parameter file.
Coordinates	=Cartesian	
	=Cylindrical	
CoreWidth	=<float>	! μm Square aperture half-width d (for Cartesian) or the circular aperture radius R (for Cylindrical).

Table 187 EffectiveIndex() Effective Index Method for VCSELs on page 777

DiffractionLoss	=(<float>...)	0 cm^{-1} Diffraction loss. Alternative to DiffractionRate.
DiffractionRate	=(<float>...)	0 ps^{-1} Diffraction rate. Alternative to DiffractionLoss.
mModeIndex	=(<int>...)	0 Index m of the parameter specified with ModeType (cylindrical harmonic order if Coordinates=Cylindrical).
ModeNumber	=<int>	1 Number of modes to be calculated.
ModeType	=(<mt>...)	Mode types: HEmn, EHmn, TEOn, or TM0n.
nModeIndex	=(<int>...)	0 Index n of the parameter specified with ModeType.
OpticalDecayRate	=(<float>...)	0 ps^{-1} Optical loss.
PowerCouplingFactor	=(<float>...)	Scaling factor for optical output power in VCSELs. VCSEL Output Power on page 714
TargetWavelength	=(<float>...)	! nm Target value for lasing wavelength.

Table 188 EmissionTable Computing Tabulated Optical Emission Data on page 844

CalibType	=FullTable	Compute full table (default).
	=nEqualp	Compute dephasing rates or tabulated data for $n = p$ only.
eDensitySweep(List (FileName=<string>)	Alternative specification of parameterization using file.
	Intervals=<int>	Number of intervals.
	Range=(<float>*2)	Parameterization range.
	Type=Lin	Linear spacing of parameterization.
	Type=Log)	Logarithmic spacing of parameterization
FrequencySweep(List (FileName=<string>)	Alternative specification of parameterization using file.
	Intervals=<int>	Number of intervals.
	Range=(<float>*2)	Parameterization range.
	Type=Lin	Linear spacing of parameterization.
	Type=Log)	Logarithmic spacing of parameterization

G: Command File Overview

Physics

Table 188 EmissionTable Computing Tabulated Optical Emission Data on page 844

hDensitySweep(List (FileName=<string>)	Alternative specification of parameterization using file.
	Intervals=<int>	Number of intervals.
	Range=(<float>*2)	Parameterization range.
	Type=Lin	Linear spacing of parameterization.
	Type=Log)	Logarithmic spacing of parameterization
NumberOfCB	=<int>	Number of electron subbands that are used for stimulated and spontaneous emission computation using the second Born approximation.
NumberOfVB	=<int>	Number of hole subbands that are used for stimulated and spontaneous emission computation using the second Born approximation.
TemperatureSweep(List (FileName=<string>)	Alternative specification of parameterization using file.
	Intervals=<int>	Number of intervals.
	Range=(<float>*2)	Parameterization range.
	Type=Lin	Linear spacing of parameterization.
	Type=Log)	Logarithmic spacing of parameterization.

Table 189 FEScalar() FE Scalar Solver on page 764

Absorption	(ODB)	Use absorption from the TableODB section of the parameter file.
Boundary	=(<type>...)	Boundary condition types, "Type1" or "Type2".
EquationType	=Waveguide	Solve waveguide problem.
LasingWavelength	=<float>	! nm Lasing wavelength.
ModeNumber	=<int>	1 Number of modes to solve.
Polarization	=(<pt>...)	Polarization type, TE (default) or TM.
PowerCouplingFactor	=(<float>...)	Scaling factor for optical output power in VCSELs. VCSEL Output Power on page 714

Table 189 FEScalar() FE Scalar Solver on page 764

Symmetry	=NonSymmetric	Boundary Conditions and Symmetry for Optical Solvers on page 769
	=Periodic	Boundary Conditions and Symmetry for Optical Solvers on page 769
	=Symmetric	* Boundary Conditions and Symmetry for Optical Solvers on page 769
TargetEffectiveIndex	=(<float>...)	! Initial guess for effective refractive index.
TargetLoss	=(<float>...)	0 cm ⁻¹ Initial guess for propagation loss.

Table 190 FEVectorial() FE Vectorial Solver on page 765

Absorption	(ODB)	Use absorption from the TableODB section of the parameter file.
AzimuthalExpansion	=(<int>...)	! Cylindrical harmonic order m , for cylindrical cavity problems only.
Boundary	=(<type>...)	Boundary condition types, "Type1" or "Type2".
Coordinates	=Cartesian	
	=Cylindrical	For cavity problems only.
EquationType	=Cavity	Solve cavity problem.
	=Waveguide	Solve waveguide problem.
LasingWavelength	=<float>	! nm Lasing wavelength, for Waveguide only.
LongitudinalWavevector	=<float>	! m ⁻¹ Longitudinal wavevector $2\pi/\lambda_z$, for Cartesian cavity problems only.
ModeNumber	=<int>	1 Number of modes to solve.
OpticalDecayRate	=(<float>...)	0 ps ⁻¹ Additional optical loss; for cavity problems only.
PowerCouplingFactor	=(<float>...)	Scaling factor for optical output power in VCSELs. VCSEL Output Power on page 714
Symmetry	=NonSymmetric	Boundary Conditions and Symmetry for Optical Solvers on page 769
	=Periodic	Boundary Conditions and Symmetry for Optical Solvers on page 769
	=Symmetric	* Boundary Conditions and Symmetry for Optical Solvers on page 769
TargetEffectiveIndex	=(<float>...)	! Initial guess for effective refractive index, for Waveguide only.

G: Command File Overview

Physics

Table 190 FEVectorial() FE Vectorial Solver on page 765

TargetLifetime	=(<float>...)	! ps Target value for photon lifetime; for Cavity problems only.
TargetLoss	=(<float>...)	0 cm ⁻¹ Initial guess for propagation loss; for Waveguide only.
TargetWavelength	=(<float>...)	! nm Target value for lasing wavelength; for Cavity problems only.

Table 191 FullPhotonRecycle() in LED(Optics(RayTrace())) Syntax for Full Photon-Recycling Model on page 746

EstimateAbsFactor	=<[0,1]>	0 Initial estimate of absorption factor.
EstimateASEFactor	=<[1,>	1 . 0 Initial estimate of ASE factor.
PhotonRecyclingScaling	=<float>	1 . 0 Scaling factor for all photon-recycling effects.
SpectrumEnergySpan	=<float>	1 . 0 eV Energy span of spontaneous spectrum.
SpectrumNumberOfPoints	=<int>	40 Number of points to represent the spectrum during evolution.
StartEnergy	=<float>	eV User-defined starting energy of spontaneous spectrum.

Table 192 Optics() in Laser() and LED() Chapter 29 on page 761

DeterministicProblem(Table 186)	Automatic Optical Mode Searching on page 797
EffectiveIndex(Table 187)	Effective index solver. Effective Index Method for VCSELs on page 777
FEScalar(Table 189)	Scalar finite-element solver. FE Scalar Solver on page 764
FEVectorial(Table 190)	Vectorial finite-element solver. FE Vectorial Solver on page 765
OptConfin	=<float>	Optical confinement factor. Waveguide Optical Modes and Fabry–Perot Cavity on page 703
OpticsFromFile(Table 193)	Optical mode properties. Loading Optical Modes on page 784
RayTrace(Table 196)	Raytracing. Far Field on page 786
TMM1D(Table 197)	Transmission mode method. Transfer Matrix Method for VCSELs on page 775

Table 193 OpticsFromFile() [Loading Optical Modes on page 784](#)

LasingWavelength	=(<float>...)	nm Lasing wavelength, for edge-emitting laser only.
ModeNumber	=<int>	1 Number of modes to load.
OutcouplingLoss	=(<float>...)	ps ⁻¹ Decay rate through top mirror, for VCSEL only.
TargetEffectiveIndex	=(<float>...)	1 Real part of the complex propagation constant, for edge-emitting laser only.
TargetLoss	=(<float>...)	1 Imaginary part of the complex propagation constant, for edge-emitting laser. ps ⁻¹ Total decay rate, for VCSEL.
TargetWavelength	=(<float>...)	nm Lasing wavelength, for VCSEL only.

Table 194 PlotEvolvingSpectra() in LED(Optics(RayTrace())) [Plotting Evolving Spectra on page 748](#)

NumberOfPoints	=<int>	Number of discretized points used to plot the spectrum.
PlotAverage	()	Plot the volume average modified spontaneous density.
PlotLevel	=<int>	Number of photon-recycling evolutions to plot.
PlotRays	(<int>...)	List of starting ray indices. The evolving spectra will be plotted for each starting ray index.
Range	(<float>*2)	eV Energy range of evolving spectra to be printed.

Table 195 PrintORArays() in LED(Optics(RayTrace())) [Interface to LightTools® on page 754](#)

FileName	=<string>	File name used to print the ray and spectrum data.
NumberOfSpectralPoints	=<int>	Number of discretized points to use for printing the spectrum.
RangeOfSpectrum	=(<float>*2)	eV Energy range of the spectrum to be printed.
RefractiveIndexDelta	=<float>	0 . 2 Expected range of refractive index change due to spectrum band width.
SplitSpectrumRays	=<int>	Number of extra rays into which each far-field ray will be split.
VectorUnits	=Millimeter =Micron	Output position vector of ray in millimeters. Output position vector of ray in micrometers.

G: Command File Overview

Physics

Table 196 RayTrace() in LED(Optics()) [LED Raytracing on page 724](#)

ActivateSpectralConversion()	Table 183	Activate the spectral conversion feature. Spectral Conversion on page 749
Coordinates	=Cartesian	*
	=Cylindrical	
DebugLEDRadiation	(<string> <float1> <float2> <float3>)	off Trace the origin of the output rays that are within the angles [<float1>, <float2>] (in degrees) and of minimum intensity specified by the <float3> parameter. In 2D, the angle is defined in the regular polar coordinates. In 3D, the angles are defined from the z -axis, as in θ in regular spherical coordinates. The results are output to the file specified by <string>.
DepthLimit	=<int>	5 Stop tracing the ray after passing through more than <int> material boundaries.
Disable		off Disable raytracing but still run LED simulation.
EmissionType	(Anisotropic(Sine(<float>*3) Cosine(<float>*3))	
	(Isotropic)	*
ExcludeHorizontalSource	(<float>)	off Omit the source rays that are emitted within the horizontal angular zone specified by the <float> parameter (in degrees).
FullPhotonRecycle()	Table 191	Full photon-recycling model. Syntax for Full Photon-Recycling Model on page 746
LEDRadiationPara	(<float>, <int>)	μm Observation radius and discretization of the observation circle or sphere.
LEDSpectrum	(<float>*2 <int>)	eV Starting and ending energy range, and number of subdivisions in that energy range.
MinIntensity	=<float>	$1e-5$ Stop tracing the ray when the intensity of a ray becomes less than <float> times the original intensity.
MonteCarlo		Activate Monte Carlo raytracing.
MoveBoundaryStartRays	= <float>	0 nm. Shift the starting ray position at device boundary inwards. Recommended values are 1 to 5 nm.
NonActiveAbsorptionOff		Do not add nonactive region absorption as generation rate to continuity equation. Nonactive Region Absorption (Photon Recycling) on page 736

Table 196 RayTrace() in LED(Optics()) [LED Raytracing on page 724](#)

ObservationCenter	=<vector>	Fixed observation center for LED radiation. By default, center of device.
OmitReflectedRays		Discard all reflected rays in raytracing process.
OmitWeakerRays		Discard the weaker ray at a material interface. This is decided by comparing the reflectivity and transmittivity.
OptGenScaling	= <float>	Set a multiplication factor to the nonactive optical generation computed.
PlotEvolvingSpectra(Table 194	Plot the evolving spectra of the full photon-recycling feature. Plotting Evolving Spectra on page 748
PolarizationVector	= Random	Default. Automatically choose a random polarization vector that is perpendicular to the starting ray direction.
	= <vector>	Set a user-defined polarization vector.
Print		off Output all rays to a grid file.
	(ActiveVertex(<int>))	off Output only ray paths emitted from this active vertex.
	(Skip(<int>))	1 Output every other <int> ray to a grid file.
PrintORArrays(Table 195	Print ray and spectrum data into the input format of <i>LightTools®</i> for LED packaging design. Interface to LightTools® on page 754
PrintRayInfo	(<string>)	off Print all indices and positions of starting rays into the file specified by <string>. Plotting Evolving Spectra on page 748
PrintSourceVertices	(<string>)	off Print the index and coordinates of all active vertices into the file specified by <string>.
ProgressMarkers	=<int>	5 Completion meter for raytracing.
RaysPerVertex	=<int>	10 Number of rays starting from each active source vertex. For 3D, the number of starting rays are constrained by 6, 18, 68, and so on. The number in the sequence is chosen such that RaysPerVertex is slightly larger or equal to it.
RaysRandomOffset		off Randomize the angular shift of the starting rays.
RedistributeStoppedRays		Distribute the total accumulated power in terminated rays back into the raytree.

G: Command File Overview

Physics

Table 196 RayTrace() in LED(Optics()) [LED Raytracing on page 724](#)

RefractiveIndex	(model=<string>)	Use PMI model <string>. Refractive Index on page 1000
	(model=ODB)	Use TableODB parameter section.
	(model=Parameter)	* Read from parameter file.
SemAbsorption	(model=<string>)	Use PMI model <string>. Optical Absorption on page 998
	(model=ODB)	Use TableODB parameter section.
	(model=Parameter)	* Read from parameter file.
SimplePhotonRecycle	(LumpActivePerEdge)	For photon recycling in bulk active regions in 2D. Active-Region Photon Recycling on page 738
	(LumpActivePerFace)	For photon recycling in bulk active regions in 3D. Active-Region Photon Recycling on page 738
SourceRaysFromFile (<string>)	Import a set of starting ray directions from the file name specified by <string>. The number of imported rays are specified by RaysPerVertex.
Symmetry	=NonSymmetric	*
	=Symmetric	
TraceSource	()	Retrace the source of the output rays to produce a map of the source regions that give the strongest ray output.
TurnOffTreeNodeCount		Disable counting the total number of nodes in the raytree.
Wavelength	=<float>	nm Wavelength, by default, peak of spontaneous gain spectrum.

Table 197 TMM1D() [Transfer Matrix Method for VCSELs on page 775](#)

Absorption	(ODB)	Use absorption from the TableODB section of the parameter file.
PowerCouplingFactor	=<float>	Scaling factor for optical output power in VCSELs. VCSEL Output Power on page 714
SigmaGauss	=<float>	μm Width of the Gaussian distribution.
TargetWavelength	=<float>	nm Target value for lasing wavelength. Only one value can be input because TMM1D solves the fundamental mode only.

Mobility

Table 198 Mobility(), eMobility(), hMobility() [Chapter 8 on page 267](#)

CarrierCarrierScattering	(BrooksHerring)	off Use Brooks–Herring carrier–carrier scattering model. Carrier–Carrier Scattering on page 288
	(ConwellWeisskopf)	off Use Conwell–Weisskopf carrier–carrier scattering model. Carrier–Carrier Scattering on page 288
DopingDependence	(<string>)	off PMI model <string>. Doping-dependent Mobility on page 942
	(Arora)	off Use Arora doping-dependent mobility model. Doping-dependent Mobility Degradation on page 268
	(Masetti)	off Use Masetti doping-dependent mobility model. Doping-dependent Mobility Degradation on page 268
	(PMIModel (Table 220))	off Use PMI for MSC-dependent mobility. Multistate Configuration-dependent Bulk Mobility on page 947
	(UniBo)	off Use University of Bologna doping-dependent mobility model. Doping-dependent Mobility Degradation on page 268
eHighFieldSaturation(Table 199)	off Electron high-field saturation. High-Field Saturation on page 294
Enormal	(<string>)	off PMI model <string>. Mobility Degradation at Interfaces on page 949
	(IALMob)	off Inversion and accumulation layer mobility model. Mobility Degradation at Interfaces on page 273
	(Lombardi)	off Enhanced Lombardi model. Mobility Degradation at Interfaces on page 273
	(Lombardi_highk)	off Enhanced Lombardi model with high-k degradation. Mobility Degradation at Interfaces on page 273
	(UniBo)	off University of Bologna surface mobility model. Mobility Degradation at Interfaces on page 273
hHighFieldSaturation(Table 199)	off Hole high-field saturation. High-Field Saturation on page 294
HighFieldSaturation(Table 199)	off High-field saturation. High-Field Saturation on page 294

G: Command File Overview

Physics

Table 198 Mobility(), eMobility(), hMobility() Chapter 8 on page 267

IncompleteIonization		off Incomplete ionization-dependent mobility. Incomplete Ionization–dependent Mobility Models on page 302
PhuMob (off Philips unified mobility model. Philips Unified Mobility Model on page 289
	Arsenic	* Use arsenic parameters. Table 51 on page 293
	Klaassen	* Use Klaassen parameters. Table 50 on page 293
	Meyer	Use Meyer parameters. Table 50 on page 293
	Phosphorus)	Use phosphorus parameters. Table 51 on page 293
ToCurrentEnormal	as Enormal	off Dependence on electric field normal to current. Mobility Degradation at Interfaces on page 273
Tunneling		- Tunneling correction to mobility. Eq. 187, p. 261

Table 199 HighFieldSaturation() High-Field Saturation on page 294

<string>	PMI model <string>. High-Field Saturation Model on page 957
CarrierTempDrive	Temperature driving force. High-Field Saturation on page 294
CarrierTempDriveBasic	Basic temperature driving force. Basic Model on page 297
CarrierTempDriveME	Meinerzhagen–Engl temperature driving force. Meinerzhagen–Engl Model on page 298
CarrierTempDrivePolynomial	Energy-dependent mobility model using an irrational polynomial. Energy-dependent Mobility on page 568
CarrierTempDriveSpline	Energy-dependent mobility model using spline interpolation. Spline Interpolation on page 570
CaugheyThomas	* Use Canali/Hänsch model. Extended Canali Model on page 295
Eparallel	Use electric field parallel to current as driving force.
EparallelToInterface	Use electric field parallel to the closest semiconductor–insulator interface as driving force.
GradQuasiFermi	* Use gradient of quasi-Fermi potential as driving force.
PFMob	Use Poole–Frenkel mobility model. Poole–Frenkel Mobility (Organic Material Mobility) on page 303
TransferredElectronEffect	Use transferred electron model. Transferred Electron Model on page 297

Radiation Models

Table 200 [AlphaParticle\(\)](#) [Alpha Particles on page 490](#)

Direction	=<vector>	! Direction of motion of the particle.
Energy	=<float>	! MeV Energy of the alpha particle.
Location	=<vector>	! μm Point where the alpha particle enters the device (bidirectional track).
StartPoint	=<vector>	! μm Point where the alpha particle enters the device (one-directional track).
Time	=<float>	! s Time at which the charge generation peaks.

Table 201 [HeavyIon\(\)](#) [Heavy Ions on page 492](#)

Direction	=<vector>	Direction of motion of the ion.
Exponential		* Use exponential shape for the spatial distribution $R(w)$.
Gaussian		Use Gaussian shape for the spatial distribution $R(w)$.
Length	= [<float>, ...]	Length l where <code>Wt_hi</code> and <code>Let_f</code> are specified (in cm by default or μm if the <code>PicoCoulomb</code> option is selected).
LET_f	= [<float>, ...]	Linear energy transfer (LET) function (in $\text{pairs}\text{cm}^{-3}$ by default or $\text{pC}\mu\text{m}^{-1}$ if the <code>PicoCoulomb</code> option is selected). Entries in the list match those in <code>Length</code> .
Location	=<vector>	μm Point where the heavy ion enters the device (bidirectional track).
PicoCoulomb		Switch units for <code>LET_f</code> , <code>Wt_hi</code> , and <code>Length</code> .
StartPoint	=<vector>	μm Point where the heavy ion enters the device (one-directional track).
Time	=<float>	s Time at which the ion penetrates the device.
Wt_hi	= [<float>, ...]	Characteristic distance, $w_t(l)$ (in cm by default or μm if the <code>PicoCoulomb</code> option is selected). Entries in the list match those in <code>Length</code> .

G: Command File Overview

Physics

Table 202 Radiation() [Chapter 15 on page 489](#)

Dose	=<float>	rad Total dose over exposure time.
DoseRate	=<float>	rad s^{-1} Dose rate.
DoseTime	=(<float>*2)	! s Exposure time.
DoseTSigma	=<float>	! s Standard deviation of rise and fall of dose rate over time.

Various

Table 203 Aniso() [Chapter 21 on page 575](#)

eAvalanche		Use anisotropic electron avalanche generation. Anisotropic Avalanche Generation on page 583
eMobility		Use self-consistent anisotropic mobility for electrons. Self-Consistent Anisotropic Mobility on page 581
eMobilityFactor	(Total)=<float>	Total anisotropic mobility factor for electrons. Total Anisotropic Mobility on page 580
	(TotalDD)=<float>	Total direction-dependent mobility factor for electrons. Total Direction-dependent Anisotropic Mobility on page 580
hAvalanche		Use anisotropic hole avalanche generation. Anisotropic Avalanche Generation on page 583
hMobility		Use self-consistent anisotropic mobility for holes. Self-Consistent Anisotropic Mobility on page 581
hMobilityFactor	(Total)=<float>	Total anisotropic mobility factor for holes. Total Anisotropic Mobility on page 580
	(TotalDD)=<float>	Total direction-dependent mobility factor for holes. Total Direction-dependent Anisotropic Mobility on page 580
Poisson		Use anisotropic electrical permittivity. Anisotropic Electrical Permittivity on page 584
Temperature		Use anisotropic thermal conductivity. Anisotropic Thermal Conductivity on page 586

Table 204 BarrierType() in SingletExciton()

Bandgap		* (i) Use bandgap difference as barrier. Singlet Exciton Equation on page 189
CondBand		(i) Use conduction band-edge difference as barrier. Singlet Exciton Equation on page 189
ValBand		(i) Use valence band-edge difference as barrier. Singlet Exciton Equation on page 189

Table 205 Charge(...) Insulator Fixed Charges on page 363

Conc	=<float>	$q\text{cm}^{-3}$ (r) $q\text{cm}^{-2}$ (i) Oxide or interface charge density.
Gaussian		(i) Gaussian distribution.
SpaceMid	=<vector>	μm (i) Charge distribution center.
SpaceSig	=<vector>	μm (i) Charge distribution width.
Uniform		* (i) Uniform distribution.

Table 206 ComplexRefractiveIndex() Complex Refractive Index Model on page 477

CarrierDep(real	Use carrier dependency of refractive index. Carrier Dependency on page 479
	imag)	Use carrier dependency of extinction coefficient. Carrier Dependency on page 479
GainDep	(real(lin))	Use linear gain dependency of refractive index. Gain Dependency on page 479
	(real(log))	Use logarithmic gain dependency of refractive index. Gain Dependency on page 479
TemperatureDep(real)	Use temperature dependency of refractive index. Temperature Dependency on page 478
WavelengthDep(real	Use wavelength dependency of refractive index. Wavelength Dependency on page 478
	imag)	Use wavelength dependency of extinction coefficient. Wavelength Dependency on page 478

G: Command File Overview

Physics

Table 207 EffectiveIntrinsicDensity() [Band Gap and Electron Affinity on page 225](#)

BandGap	(<string>)	Use PMI model <string> to compute band gap E_g . Band Gap on page 965
BandGapNarrowing	(<string>)	Use PMI model <string> for bandgap narrowing. Bandgap Narrowing on page 968
	(BennettWilson)	* Bennett–Wilson model. Band Gap and Electron Affinity on page 225
	(delAlamo)	del Alamo model. Band Gap and Electron Affinity on page 225
	(JainRoulston)	Jain–Roulston model. Band Gap and Electron Affinity on page 225
	(oldSlotboom)	Old Slotboom model. Band Gap and Electron Affinity on page 225
	(Slotboom)	Slotboom model. Band Gap and Electron Affinity on page 225
	(TableBGN)	Table-based model. Band Gap and Electron Affinity on page 225
NoBandGapNarrowing		No bandgap narrowing. Band Gap and Electron Affinity on page 225
NoFermi		off Omit correction Eq. 149 even when using Fermi statistics. Bandgap Narrowing with Fermi Statistics on page 235

Table 208 eQuantumPotential() and hQuantumPotential() [Density Gradient Quantization Model on page 260](#)

Density		- (r) Use Eq. 184, p. 260 instead of Eq. 185, p. 260.
Ignore		- (r) Compute, but do not apply quantum correction.
LocalModel	=<string>	(r) Name of PMI for apparent band-edge shift. Chapter 7 on page 251, Apparent Band-Edge Shift on page 970
	=SchenkBGN_elec	(r) Schenk bandgap narrowing model for electrons. Schenk Bandgap Narrowing Model on page 231
	=SchenkBGN_hole	(r) Schenk bandgap narrowing model for holes. Schenk Bandgap Narrowing Model on page 231
Resolve		- (r) Use more accurate discretization of non-heterointerfaces.

Table 209 HeatCapacity() [Heat Capacity on page 647](#)

<string>		PMI model <string> for lattice heat capacity. Heat Capacity on page 994
Constant		Constant lattice heat capacity. Heat Capacity on page 647
PMIModel (Table 220)	Use multistate configuration-dependent model.
TempDep		* Temperature-dependent lattice heat capacity. Heat Capacity on page 647

NOTE In Table 210, d is 3 for bulk reactions and 2 for interface reactions.

Table 210 HydrogenReaction() [Reactions Between Mobile Elements on page 381](#)

FieldFromMaterial	=<string>	(i) Material where the electric field is obtained.
FieldFromRegion	=<string>	(i) Region where the electric field is obtained.
ForwardReactionCoef	=<float>	0 /cm ^d s Forward reaction coefficient.
ForwardReactionEnergy	=<float>	0 eV Forward reaction activation energy.
ForwardReactionFieldCoef	=<float>	0 cm/V Forward reaction field coefficient.
LHSCoef (Define particle numbers to be removed.
	Electron=<int>	0 Number of electrons to be removed.
	Hole=<int>	0 Number of holes to be removed.
	HydrogenAtom=<int>	0 Number of hydrogen atoms to be removed.
	HydrogenIon=<int>	0 Number of hydrogen ions to be removed.
	HydrogenMolecule=<int>	0 Number of hydrogen molecules to be removed.
Material	=<string>	(i) Material where the recombination rate enters.
Region	=<string>	(i) Region where the recombination rate enters.
ReverseReactionCoef	=<float>	0 /cm ^d s Reverse reaction coefficient.
ReverseReactionEnergy	=<float>	0 eV Reverse reaction activation energy.
ReverseReactionFieldCoef	=<float>	0 cm/V Reverse reaction field coefficient.

G: Command File Overview

Physics

Table 210 HydrogenReaction() Reactions Between Mobile Elements on page 381

RHSCoef (Define particle numbers to be created.
	Electron=<int>	0 Number of electrons to be created.
	Hole=<int>	0 Number of holes to be created.
	HydrogenAtom=<int>	0 Number of hydrogen atoms to be created.
	HydrogenIon=<int>	0 Number of hydrogen ions to be created.
	HydrogenMolecule=<int>)	0 Number of hydrogen molecules to be created.

Table 211 Incompletelionization() Chapter 6 on page 245

Dopants	=<string>	Restrict incomplete ionization to dopants named in <string>. Dopant names are separated by spaces.
Model (<string1>(<string2>)	Use PMI model <string1> for species named in <string2>. Incomplete Ionization on page 1015

Table 212 GateCurrent() Chapter 17 on page 511, Chapter 18 on page 531

<string>(<carrier>...)	(i) Use PMI model <string> to compute hot-carrier injection. Hot-Carrier Injection on page 1022
DirectTunneling		(i) Schenk direct tunneling model. Direct Tunneling on page 514
eFiegna		(i) Fiegna model for electrons. Fiegna Hot-Carrier Injection on page 536
eLucky		(i) Lucky electron model. Classical Lucky Electron Injection on page 535
Fowler		(i) Fowler–Nordheim tunneling. Fowler–Nordheim Tunneling on page 512
	(EVB)	(i) Fowler–Nordheim valence band tunneling of electrons. Fowler–Nordheim Tunneling on page 512
GateName	=<string>	(i) Electrode for monitoring gate current. Overview on page 531
hFiegna		(i) Fiegna model for holes. Fiegna Hot-Carrier Injection on page 536
hLucky		(i) Lucky hole model. Classical Lucky Electron Injection on page 535

Table 212 `GateCurrent()` [Chapter 17 on page 511](#), [Chapter 18 on page 531](#)

Interface		(i) Activate carrier injection with explicitly evaluated boundary conditions for continuity equations during a transient (Carrier Injection with Explicitly Evaluated Boundary Conditions for Continuity Equations on page 545) or if not in transient monitor interface current. Overview on page 531
TailDistribution		(i) Tail distribution model for electrons. Tail Distribution Hot-Carrier Injection on page 537
TailhDistribution		(i) Tail distribution model for holes. Tail Distribution Hot-Carrier Injection on page 537

Table 213 `Model()`, options of stress-dependent models [Chapter 23 on page 595](#)

Deformation(Linear deformation potential model for computing band structure. Deformation of Band Structure on page 596
	Minimum	Compute conduction and valence band edges using minimum band energies. Deformation of Band Structure on page 596
	ekp	k · p method for electron bands to account for shear strain components. Deformation of Band Structure on page 596
	hkp)	6x6 k · p method for hole bands. Deformation of Band Structure on page 596
DOS(eMass	Strained electron effective mass and DOS model. Strained Electron Effective Mass and DOS on page 601
	hMass	Strained hole effective mass and DOS model. Strained Hole Effective Mass and DOS on page 603
	hMass (AnalyticLTFit)	Strained hole effective mass and DOS model with analytic lattice temperature fit. Strained Hole Effective Mass and DOS on page 603
	hMass (NumericalIntegration))	Strained hole effective mass and DOS model with numeric integrations. Strained Hole Effective Mass and DOS on page 603

G: Command File Overview

Physics

Table 213 Model(), options of stress-dependent models [Chapter 23 on page 595](#)

Mobility(eFactor	Piezoresistive factor for electrons. Piezoresistance Mobility Factor Models on page 615
	eFactor(Kanda)	Piezoresistive factor for electrons with doping dependency. Piezoresistance Mobility Factor Models on page 615
	eSubband	Strain-induced subband model for electrons. Stress-induced Electron Mobility Model on page 617
	eSubband(EffectiveMass)	Stress-induced change of the electron effective mass. Effective Mass on page 621
	eSubband(Doping)	Strain-induced subband model for electrons with doping dependency. Stress-induced Electron Mobility Model on page 617
	eSubband(Fermi)	Strain-induced subband model for electrons with carrier concentration (Fermi statistics) dependency. Stress-induced Electron Mobility Model on page 617
	eSubband(Scattering)	Strain-induced subband model for electrons with scattering. Intervalley Scattering on page 620
	eSubband(Enormal)	Strain-induced subband model for electrons with an experimental option that tries to account for a quantization in MOSFET channel. Stress-induced Electron Mobility Model on page 617
	eTensor	Piezoresistive tensor for electrons. Piezoresistance Mobility Model on page 606
	eTensor(Kanda)	Piezoresistive tensor for electrons with doping dependence. Piezoresistance Mobility Model on page 606
	eMinorityFactor=<float>	Factor to scale stress effect for minority electrons. Piezoelectric Polarization on page 635
	eMinorityFactor(ThresholdDoping=<float>)=<float>	Factor to scale stress effect for minority electrons with no doping dependence. Piezoelectric Polarization on page 635
	hFactor	Piezoresistive factor for holes. Piezoresistance Mobility Factor Models on page 615
	hFactor(Kanda)	Piezoresistive factor for holes with doping dependency. Piezoresistance Mobility Factor Models on page 615
	hSixband	Intel stress-induced model for holes. Intel Stress-induced Hole Mobility Model on page 624

Table 213 Model(), options of stress-dependent models [Chapter 23 on page 595](#)

Mobility(hSixband(Doping)	Intel stress-induced model for holes with doping dependency. Intel Stress-induced Hole Mobility Model on page 624
	hSixband(Fermi)	Intel stress-induced model for holes with carrier concentration (Fermi statistics) dependency. Intel Stress-induced Hole Mobility Model on page 624
	hTensor	Piezoresistive tensor for holes. Piezoresistance Mobility Model on page 606
	hTensor(Kanda))	Piezoresistive tensor for holes with doping dependency. Piezoresistance Mobility Model on page 606
	hMinorityFactor=<float>	Factor to scale stress effect for minority holes. Piezoelectric Polarization on page 635
	hMinorityFactor(ThresholdDoping=<float>)=<float>	Factor to scale stress effect for minority holes with no doping dependency. Piezoelectric Polarization on page 635

Table 214 MoleFraction() [Mole-Fraction Specification on page 550](#)

Grading()		Alternative way to specify grading; allows a nonzero mole fraction and different distance of grading from different parts of the boundaries.
	GrDistance=<float>	μm Distance in the direction normal to the specified interface, where linear interpolation of mole fraction(s) from the constant value to the specified boundary mole fraction(s) occurs.
	RegionInterface=<string>*2)	Restrict this grading to the given interface (applied to all interfaces by default).
	xFraction=<[0,1]>	Boundary xMoleFraction.
	yFraction=<[0,1]>...)	Boundary yMoleFraction.
GrDistance	=<float>	μm Distance in the direction normal to the boundaries of the specified region(s) where linear interpolation of mole fraction(s) from the specified constant value to 0 occurs.
RegionName	=<string>...]	List of regions where the mole-fraction specification will take effect.
	=<string>	Regions where the mole-fraction specification will take effect.
xFraction	=<[0,1]>	Constant value of xMoleFraction.
yFraction	=<[0,1]>	Constant value of yMoleFraction.

G: Command File Overview

Physics

Table 215 MSConfig() [Chapter 11 on page 365](#)

BandEdgeShift(<string> [<int>])	Compute band-edge shifts for conduction and valence by PMI model <string> with optional constructor argument <int>. Apparent Band-Edge Shift on page 368
Conc	=<float>	0 cm^{-d} Concentration of states ($d = 3$ for bulk; $d = 2$ for interface MSCs).
eBandEdgeShift(<string> [<int>])	Compute band-edge shift for conduction band by PMI model <string> with optional constructor argument <int>. Apparent Band-Edge Shift on page 368
Elimination		+ Switch solving algorithm.
hBandEdgeShift(<string> [<int>])	Compute band-edge shift for valence band by PMI model <string> with optional constructor argument <int>. Apparent Band-Edge Shift on page 368
Name	=<string>	! Identifier of MSConfig.
State(! Define a state (at least two required).
	Charge=<int>	0 Number of positive elementary charges.
	Hydrogen=<int>	0 Number of hydrogen atoms.
	Name=<string>)	! Identifier of state.
Transition(Table 223)	! Define a transition.

Table 216 Noise() [Chapter 16 on page 499](#)

DiffusionNoise(e_h_Temperature	Diffusion noise; noise temperatures are the respective carrier temperatures. Diffusion Noise on page 501
	eTemperature	Diffusion noise; noise temperature is electron temperature for electrons, lattice temperature for holes. Diffusion Noise on page 501
	hTemperature	Diffusion noise; noise temperature is lattice temperature for electrons, hole temperature for holes. Diffusion Noise on page 501
	LatticeTemperature)	* Diffusion noise; noise temperature is lattice temperature. Diffusion Noise on page 501
Doping(BandGapNarrowing	Random dopant fluctuations, including their effect on bandgap narrowing. Random Dopant Fluctuations on page 503
	Mobility)	Random dopant fluctuations, including their effect on mobility. Random Dopant Fluctuations on page 503
FlickerGRNoise		Bulk flicker noise. Bulk Flicker Noise on page 502

Table 216 Noise() [Chapter 16 on page 499](#)

MonopolarGRNoise		Equivalent monopolar noise. Equivalent Monopolar Generation–Recombination Noise on page 502
Traps		Trapping noise. Trapping Noise on page 503

Table 217 Optics() [Transfer Matrix Method on page 438](#) and [Beam Propagation Method on page 462](#)

Table 192		Optics stand-alone. Optics Stand-alone Option on page 795
BPMScalar (Table 164)	Scalar beam propagation method (BPM) solver. Beam Propagation Method on page 462
TMM (Table 180)	Transfer matrix method (TMM) solver. Transfer Matrix Method on page 438

Table 218 Optics() [Unified Interface for Optical Generation Computation on page 393](#)

ComplexRefractiveIndex (Table 206)	Complex refractive index models. Complex Refractive Index Model on page 477
Excitation (Specification of excitation parameters. Setting the Excitation Parameters on page 404
	Intensity=<float>	Wcm ⁻²
	Phi=<float>	0 deg Angle between projection of propagation direction on xy plane and x-axis.
	Polarization=<float>	[0,1] Definition of polarization as in Using Transfer Matrix Method on page 442 .
	Polarization=<ident>	Transverse electric (TE) or transverse magnetic (TM) polarized light.
	PolarizationAngle=<float>	deg Angle between H-field and $\hat{z} \times \hat{k}$ used for vectorial solvers only.
	Theta=<float>	0 deg Angle between propagation direction and z-axis.
	Wavelength=<float>	μm
OpticalGeneration (Optical generation models. Specifying the Type of Optical Generation Computation on page 394

G: Command File Overview

Physics

Table 218 Optics() Unified Interface for Optical Generation Computation on page 393

OpticalSolver (Specification of optical solver method. Specifying the Optical Solver on page 400
	FDTD ()	
	TMM (Table 180)	See Using Transfer Matrix Method on page 442 . Note that Excitation section must not be defined in TMM section. Unified Interface for Optical Generation Computation on page 393

Table 219 Piezo() Chapter 23 on page 595

Model (Table 213)	Stress-dependent models. Deformation of Band Structure on page 596 to Intel Stress-induced Hole Mobility Model on page 624
OriKddX	=<vector>	(1, 0, 0) Miller indices of the stress system relative to the simulation system.
OriKddY	=<vector>	(0, 1, 0) Miller indices of the stress system relative to the simulation system.
Stress	=(<float>*6)	Pa Components xx, yy, zz, yz, xz, xy of stress tensor.
	=<string>	Use PMI model <string> to compute stress. Stress on page 1002

Table 220 PMIModel() Multistate Configuration–dependent Bulk Mobility on page 947, Multistate Configuration–dependent Thermal Conductivity on page 991, Multistate Configuration–dependent Heat Capacity on page 996

Index	=<int>	0 Number passed to and interpreted by PMI model.
MSConfig	=<string>	" " Name of multistate configuration on which PMI depends.
Name	=<string>	! Name of PMI model.
String	=<string>	" " String passed to and interpreted by PMI model.

Table 221 Schroedinger() 1D Schrödinger Solver on page 253

DensityTail	=Extrapolate	* Use estimate for lowest noncomputed subband energy as $E_{\max, v}$. Eq. 182, p. 259
	=MaxEnergy	Use highest computed subband energy as $E_{\max, v}$. Eq. 182, p. 259
Electron		- Solve Schrödinger equation for electrons.
EnergyInterval	[(<carrier>)]=<float>	0 eV Highest energy to which eigensolutions are computed, measured from the lowest interior potential point on the nonlocal line on which the Schrödinger equation is solved. When 0, only bound solutions are computed.
Error	[(<carrier>)]=<(0,)>	1e-5 eV Precision target for eigenenergies.
Hole		- Solve Schrödinger equation for holes.
MaxSolutions	[(<carrier>)]=<0..>	5 Maximum number of eigensolutions computed per ladder.
Smooth	=<float>	0 cm Length (measured from end of nonlocal line) over which to blend from classical to quantum density.

Table 222 ThermalConductivity() Thermal Conductivity on page 648

<string>		Use PMI model <string> for thermal conductivity. Thermal Conductivity on page 989
Constant	Conductivity	Use constant conductivity.
	Resistivity	Use constant resistivity.
Formula		Use the built-in strategy for thermal conductivity. Thermal Conductivity on page 989
PMIModel(Table 220)	Use multistate configuration-dependent model.
TempDep	Conductivity	Use Eq. 707, p. 648 .
	Resistivity	Use Eq. 706, p. 648 .

Table 223 Transition() in MSConfig() [Chapter 11 on page 365](#)

CEModel(<string> [<int>])	! Use PMI_TrapCaptureEmission model <string> with optional constructor argument <int>.
From	=<string>	! Interacting state.
	=<string>	! Identifier of transitions.
Reservoirs(List of reservoirs for particle conservation.
	<string>(Particles =<int>) ...	Reservoir <string> and number of involved particles <int>.
To	=<string>	! Reference state.

NOTE In Table 224, $d = 3$ for bulk traps and $d = 2$ for interface traps.

Table 224 Traps(...) [Chapter 10 on page 349](#), [Chapter 12 on page 373](#)

Acceptor		Acceptor trap type. Trap Types on page 350
ActEnergy	=<float>	eV Equilibrium activation energy of hydrogen on Si-H bonds, ε_A^0 . Chapter 12 on page 373
Add2TotalDoping		Include the traps in the doping concentrations used to compute mobility, lifetimes, and so on. Material and Doping Specification on page 123
BondConc	=<float>	cm^{-d} Total silicon dangling bond concentration N . Chapter 12 on page 373
CBRate	=<string>	Use PMI <string> to compute electron capture and emission rate for conduction band.
	=(<string> <int>)	Use PMI <string> with constructor argument <int> to compute electron capture and emission rate for conduction band. Trap Capture and Emission Rates on page 1008
Conc	=<float>	cm^{-d} or $\text{eV}^{-1} \text{cm}^{-d}$ Concentration or the peak density of the trap distribution. Energetic and Spatial Distribution of Traps on page 350
CritConc	=<float>	cm^{-d} Critical concentration N_{crit} , default $0.1N$. Chapter 12 on page 373
CurrentEnhan	(<float>*4)	$(0 \ 1 \ 0 \ 1) \ \text{cm}^{2\rho_{\text{Tun}} A^{-\rho_{\text{Tun}}}}, 1, \text{cm}^{2\rho_{\text{HC}} A^{-\rho_{\text{HC}}}}, 1$ Parameters of the tunneling and hot carrier-dependent terms of the depassivation constant, δ_{Tun} , ρ_{Tun} , δ_{HC} , and ρ_{HC} . Chapter 12 on page 373
Degradation		Use degradation model based on kinetic equation. Chapter 12 on page 373
	(PowerLaw)	Use degradation model based on power law. Chapter 12 on page 373

Table 224 Traps(...) Chapter 10 on page 349, Chapter 12 on page 373

DePasCoef	=<float>	s ⁻¹ Depassivation coefficient v_0 at the passivation conditions (the equilibrium at the passivation temperature T_0). Chapter 12 on page 373
DiffusionEnhan	=(<float>*6)	cm cm ² s ⁻¹ eV cms ⁻¹ cm ⁻³ 1 Parameters of hydrogen diffusion in oxide, x_p , D_0 , ϵ_H , k_p , N_H^0 , and N_{ox} in Eq. 390, p. 375, Chapter 12 on page 373.
Donor		Donor trap type. Trap Types on page 350
eBarrierTunneling	[(NonLocal =<string>...)]	Use nonlocal tunneling from the conduction band at reference surface of all connected unnamed, and all listed, connected named nonlocal meshes. Tunneling and Traps on page 357
eConstEmissionRate	=<float>	s ⁻¹ Constant electron emission rate term to the conduction band e_{const}^n . Local Trap Capture and Emission on page 354
eGfactor	=<float>	Electron degeneracy factor g_n . Trap Occupation Dynamics on page 352
eJfactor	=<float>	Electron J-model factor g_n^J . Local Trap Capture and Emission on page 354
ElectricField	[(<carrier>)]	Field-dependent model for cross sections. J-Model Cross Sections on page 355
EnergyMid	=<float>	eV Central energy E_0 of the trap distribution. Eq. 362, p. 350
EnergyShift	=<string>	Use PMI <string> to compute trap energy shift.
	=(<string> <int>)	Use PMI <string> with constructor argument <int> to compute trap energy shift. Energetic and Spatial Distribution of Traps on page 350, Trap Energy Shift on page 1011
EnergySig	=<(0,)>	eV Width E_S of the trap distribution. Eq. 362, p. 350
eNeutral		Electron trap type. Trap Types on page 350
Exponential		Exponential energetic distribution. Energetic and Spatial Distribution of Traps on page 350, Eq. 362, p. 350
eXsection	=<[0,)>	cm ² Electron capture cross section σ_n^0 . Local Trap Capture and Emission on page 354
FieldEnhan	(<float>*4)	(0 1 0 1) eVcm ^{$\rho_{//} - \rho_{//}$} , 1, eVcm ^{$\rho_{\perp} - \rho_{\perp}$} , 1 Parameters of the electric field-dependent terms of the Si-H bond energy and the activation energy, $\delta_{//}$, $\rho_{//}$, δ_{\perp} , and ρ_{\perp} . Chapter 12 on page 373
FixedCharge		Fixed charge. Trap Types on page 350
fromCondBand		Zero point for E_0 is conduction band. Eq. 363, p. 351
fromMidBandGap		Zero point for E_0 is intrinsic energy. Eq. 363, p. 351
fromValBand		Zero point for E_0 is valence band. Eq. 363, p. 351

G: Command File Overview

Physics

Table 224 Traps(...) Chapter 10 on page 349, Chapter 12 on page 373

Gaussian		Gaussian energetic distribution. Energetic and Spatial Distribution of Traps on page 350 , Eq. 362, p. 350
hBarrierTunneling	[(NonLocal= <string>...)]	Use nonlocal tunneling from the valence band at reference surface of all connected unnamed, and all listed, connected named nonlocal meshes. Tunneling and Traps on page 357
hConstEmissionRate	=<float>	s ⁻¹ Constant hole emission rate to the valence band e_{const}^p . Local Trap Capture and Emission on page 354
hGfactor	=<float>	Hole degeneracy factor g_p . Trap Occupation Dynamics on page 352
hJfactor	=<float>	Hole J-model factor g_p^J . Local Trap Capture and Emission on page 354
hNeutral		Hole trap type. Trap Types on page 350
HuangRhys	=<[0,)>	Huang–Rhys factor. Tunneling and Traps on page 357
hXsection	=<[0,)>	cm ² Hole capture cross section σ_p^0 . Local Trap Capture and Emission on page 354
Level		Trap with single energy level. Energetic and Spatial Distribution of Traps on page 350 , Eq. 362, p. 350
Makram-Ebeid_Lannoo	[<carrier> <simpleCapt>]	Use Makram-Ebeid–Lannoo model. Local Capture and Emission Rates Based on Makram-Ebeid–Lannoo Phonon-assisted Tunnel Ionization Model on page 356
Material	=<string>	(i) Material to which energy specification refers. Energetic and Spatial Distribution of Traps on page 350
Name	=<string>	Identifier for trap.
PasCoef	=<float>	s ⁻¹ Passivation coefficient γ_0 . By default, computed automatically to provide the equilibrium, $v_0 N_{\text{hb}}^0 / (N - N_{\text{hb}}^0)$. Chapter 12 on page 373
PasTemp	=<float>	300 K Passivation temperature T_0 . Chapter 12 on page 373
PasVolume	=<float>	0 cm ² cm ³ Passivation volume Ω . Chapter 12 on page 373
PhononEnergy	=<[0,)>	eV Phonon energy for inelastic tunneling. Tunneling and Traps on page 357
PooleFrenkel	[(<carrier>)]	Use Poole–Frenkel model. Poole–Frenkel Model for Cross Sections on page 355
PowerEnhan	(<float>*3)	(0 0 0) 1, V ⁻¹ cm, V ⁻¹ cm Parameters in the chemical potential for kinetic equation degradation and the power for degradation by power law, β_0 , β_{\parallel} , and β_{\perp} . Chapter 12 on page 373
ReferencePoint	=<vector>	Coordinate where to take the data for EnergyShift.

Table 224 Traps(...) Chapter 10 on page 349, Chapter 12 on page 373

Region	=<string>	(i) Region to which energy specification refers. Energetic and Spatial Distribution of Traps on page 350
Sfactor	=<string>	Dataset for the spatial distribution of the traps. Energetic and Spatial Distribution of Traps on page 350
	=<string>	Name of model to be used by the PMI to compute the spatial trap distribution. Trap Space Factor on page 1006
SpaceMid	=<vector>	(0 0 0) μm Center of spatial distribution. Energetic and Spatial Distribution of Traps on page 350
SpaceSig	=<vector>	(1e100 1e100 1e100) μm Width of spatial distribution. Energetic and Spatial Distribution of Traps on page 350
SpatialShape	=Gaussian	Use Gaussian shape function. Energetic and Spatial Distribution of Traps on page 350
	=Uniform	* Use Uniform shape function. Energetic and Spatial Distribution of Traps on page 350
Table	=(<float>*2...)	eV $\text{eV}^{-1}\text{cm}^{-d}$ Pairs of energy and concentration of a tabular approximation to a trap distribution. Energetic and Spatial Distribution of Traps on page 350, Eq. 362
TaileDistribution	(<float>*3)	(0 0 0) cm^2A^{-1} , eV, eV Parameters of the tail electron energy-dependent terms of the depassivation constant, δ_{tail} , ε_{th} , and ε_a . Trap Degradation Model on page 373
TailhDistribution	(<float>*3)	(0 0 0) cm^2A^{-1} , eV, eV Parameters of the tail hole energy-dependent terms of the depassivation constant, δ_{tail} , ε_{th} , and ε_a . Trap Degradation Model on page 373
TrapVolume	=<[0,)>	μm^3 Interaction volume for nonlocal tunneling. Tunneling and Traps on page 357
Tunneling(Hurkx[(<carrier>)])	Use Hurkx trap-assisted tunneling model. Hurkx Model for Cross Sections on page 355
Uniform		Uniform energetic distribution. Energetic and Spatial Distribution of Traps on page 350, Eq. 362, p. 350
VBRate	=<string>	Use PMI <string> to compute hole capture and emission rate for valence band.
	=(<string> <int>)	Use PMI <string> with constructor argument <int> to compute hole capture and emission rate for valence band. Trap Capture and Emission Rates on page 1008

Plotting

Table 225 Plot{} [Plot Section on page 58](#)

Table 121		Scalar plot data. Plot Section on page 58
Table 121/Element		Scalar plot datasets defined on elements.
Table 121/RegionInterface		Scalar plot data on region interfaces; supported for just a few datasets. Interface Plots on page 58
Table 122/Vector		Vectorial plot data. Plot Section on page 58
Table 123/SpecialVector		Special vectorial plot data; supported for the tail distribution data. Tail Distribution Hot-Carrier Injection on page 537
Table 230		Occupation rates of MSConfig states. Specifying Multistate Configurations on page 367

Table 226 Average(), Maximum(), and Minimum() [CurrentPlot Section on page 59](#)

Table 238		Sample over location.
Coordinates		Print coordinates where maximum, minimum, or average occurs.
DopingWell	<vector>	Sample over well, defined by point in the well.
Everywhere		Sample over entire device.
Insulator		Sample over all insulator regions of the device.
Name	=<string>	Name for sampled data to be used in output.
Semiconductor		Sample over all semiconductor regions of the device.
Window[<vector> <vector>]	Sample over the window, defined by two specified corners.

Table 227 CurrentPlot{} [CurrentPlot Section on page 59](#)

<int>		Print data at vertex <int>.
<vector>		μm Print data at coordinate <vertex>.
Average(Table 226	Print average over given domain.
Integrate(Table 226	Print integral over given domain.
Maximum(Table 226	Print maximum in given domain.
Minimum(Table 226	Print minimum in given domain.
Model	=<string>	Select a model for printing of parameters.
Parameter	=<string>	Print parameter <string>.

Table 228 FarfieldPlot{} [Far Field on page 786](#)

Interval	=<int>	Number of points per axis.
Range	=(<float>*2)	degree Horizontal and vertical angle ranges.
Scalar1D		* Type of far-field plot.
Scalar2D		Type of far-field plot.
Vector2D		Type of far-field plot.

Table 229 GainPlot{} [Modal Gain as Function of Energy/Wavelength on page 856](#)

Intervals	=<int>	Number of points to plot for each gain curve.
PlotLasingSpectrum		Enable plotting of laser power spectra. Plotting the Laser Power Spectrum on page 698
Range	=(<float>*2)	eV Energy range of the gain plot.

Table 230 MSConfig in Plot{} [Specifying Multistate Configurations on page 367](#)

MSConfig		All state occupations of all MSConfig.
	(<string>)	All state occupations of MSConfig <string>.
	(<string1> <string2>)	Occupation of state <string2> of MSConfig <string1>.

G: Command File Overview

Plotting

Table 231 NoisePlot{} [Noise Output Data on page 506](#)

Table 121	Scalar plot data.
Table 122 /Vector	Vector plot data.
AllLNS	All used local noise sources.
AllINVSD	All used local noise voltage spectral densities.
AllINVXVSD	All use local noise voltage cross-correlation spectral densities.
GreenFunctions	All used Green's functions and their gradients.

Table 232 NonLocalPlot{} [Visualizing Data Defined on Nonlocal Meshes on page 106](#)

Table 121		Scalar data.
EigenEnergy (Electron[(Number=<int>)] Hole[(Number=<int>)] WaveFunction as EigenEnergy		Eigenenergies. 1D Schrödinger Solver on page 253
	Electron[(Number=<int>)]	Restrict output to [<int> lowest] electron eigensolutions.
	Hole[(Number=<int>)]	Restrict output to [<int> lowest] hole eigensolutions.
WaveFunction as EigenEnergy		Wavefunctions. 1D Schrödinger Solver on page 253

Table 233 TensorPlot(){} [Visualizing Results on Native Tensor Grid on page 471](#)

Name	=<string>	! Name of tensor plot. The file name for the <code>TensorPlot</code> given in the <code>File</code> section is appended by this name.
OutputLevel	=maximum	For bidirectional BPM, tensor plots are generated not only for the final solution, but also after every iteration if maximum is specified.
Xconst	=<float>	μm X-coordinate.
Xmax	=<float>	μm Maximum x-coordinate.
Xmin	=<float>	μm Minimum x-coordinate.
Yconst	=<float>	μm Y-coordinate.
Ymax	=<float>	μm Maximum y-coordinate.
Ymin	=<float>	μm Minimum y-coordinate.

Table 233 [TensorPlot\(\){} Visualizing Results on Native Tensor Grid](#) on page 471

Zconst	=<float>	μm Z-coordinate.
Zmax	=<float>	μm Maximum z-coordinate.
Zmin	=<float>	μm Minimum z-coordinate.

Table 234 [VCSELNearFieldPlot{} VCSEL Near Field and Far Field](#) on page 793

NumberOfPoints	=<int>	Discretization of the near-field plot.
Position	=<float>	y -coordinate of the near field.
Radius	=<float>	Radius of near field.

Various

Table 235 Locations, all

Table 236		Bulk location.
Table 237		Interface location.
Electrode	=<string>	Name of an electrode that must exist in the device.

Table 236 Locations, bulk (dimension is that of the device)

Material	=<string>	Name of a material.
Region	=<string>	Name of a region that must exist in the device.

Table 237 Locations, interface (dimension is one less than that of the device)

MaterialInterface	=<string>	Material interface of the form "<ident1>/<ident2>", with <ident1> and <ident2> as the names of materials.
RegionInterface	=<string>	Region interface of the form "<ident1>/<ident2>", with <ident1> and <ident2> as the names of regions that must exist in the device and must have a common interface.

G: Command File Overview

Various

Table 238 Locations, noncontact

Table 236	Bulk location.
Table 237	Interface location.

Table 239 Optics() in Physics{}

Table 192		Optics stand-alone. Optics Stand-alone Option on page 795
BPMScalar (Table 164)	Scalar beam propagation method (BPM) solver. Beam Propagation Method on page 462
TMM (Table 164)	Transfer matrix method (TMM) solver. Transfer Matrix Method on page 438