Genius

Semiconductor Device Simulator

Version 1.7.4

Genius User's Guide

Cgenda Pte Ltd

This documentation was typed in DocBook XML format, and typeset with the Lyx program. We sincerely thank the contributors of the two projects, for their excellent work as well as their generoisty.

Contents

1 Physics in Genius Device Simulator **10**

1.1 Level 1 Drift-Diffusion Equation 10

1.2 Level 2 Drift-Diffusion Equation 12

1.3 Level 3 Energy Balance Equation 13

1.4 Band Structure Model 15

1.4.1 Band structure of compound semiconductors 17

1.4.1.1 Band Structure of SiGe 17

1.4.1.2 Band Structure of Tertiary Compound Semiconductor 17

1.4.2 Schenk's Bandgap Narrowing Model 17

1.5 Carrier Recombination 18

1.6 Mobility Models 19

1.6.1 Bulk Mobility Models 20

1.6.1.1 Analytic Mobility Model 20

1.6.1.2 Masetti Analytic Model 21

1.6.1.3 Philips Mobility Model 22

1.6.1.4 Velocity Saturation 24

1.6.2 Unified Mobility Models 25

1.6.2.1 Lombardi Surface Mobility Model 26

1.6.2.2 Lucent High Field Mobility Model 27

1.6.2.3 Hewlett-Packard High Field Mobility Model . 29
1.6.3 Mobility Models of Complex Compound Semiconductors 30
1.6.4 Carrier Temperature Based Mobility Model 30
1.7 Generation Model 30
1.7.1 Impact Ionization 30
1.7.1.1 Selberherr Model 31
1.7.1.2 van Overstraeten - de Man model 32
1.7.1.3 Valdinoci Model 33
1.7.2 Band-to-band Tunneling 34
1.8 High Energy Particles 34
1.9 Fermi-Dirac Statistics 35

2 Genius Command Reference 38

2.1 Introduction 38
2.1.1 Format of Input Card 38
2.1.2 Statement Description Format 39
2.2 Global Specification 39
2.2.1 GLOBAL 39
2.2.1.1 Parameters 40
2.2.1.2 Description 41
2.2.1.3 Example 41
2.3 Mesh Generation 41
2.3.1 MESH 42
2.3.1.1 Parameters 42
2.3.1.2 Description 43
2.3.1.3 Example 44

2.3.2 X.MESH Y.MESH and Z.MESH 44
2.3.2.1 Parameters 44
2.3.2.2 Description 45
2.3.2.3 Example 45
2.3.3 ELIMINATE 46
2.3.3.1 Parameters 46
2.3.3.2 Description 47
2.3.3.3 Example 48
2.3.4 SPREAD 48
2.3.4.1 Parameters 48
2.3.4.2 Example 49
2.3.5 SPREAD3D 49
2.3.5.1 Parameters 49
2.3.5.2 Example 51
2.3.6 REGION 51
2.3.6.1 Parameters 51
2.3.6.2 Description 52
2.3.6.3 Example 53
2.3.7 FACE 53
2.3.7.1 Parameters 53
2.3.7.2 Description 54
2.3.7.3 Example 54
2.3.8 REFINE 55
2.3.8.1 Parameters 55
2.3.8.2 Description 57
2.3.8.3 Example 58

2.4 External Sources 58

2.4.1 VSOURCE 58

2.4.1.1 Parameters 58

2.4.1.2 Description 60

2.4.1.3 Example 60

2.4.2 ISOURCE 60

2.4.2.1 Parameters 61

2.4.2.2 Example 62

2.4.3 PARTICLE 62

2.4.3.1 Parameters 63

2.4.3.2 Description 63

2.4.3.3 Examples 63

2.4.4 LIGHT 63

2.4.4.1 Parameters 64

2.4.4.2 Description 65

2.4.4.3 Examples 66

2.4.5 ENVELOP 66

2.4.5.1 Parameters 67

2.4.5.2 Examples 67

2.4.6 CIRCUIT 68

2.4.6.1 Parameters 68

2.5 Boundary Conditions 68

2.5.1 Boundary and Contact 68

2.5.1.1 Parameters 69

2.5.1.2 Description 70

2.5.1.3 Example 71

2.5.2 ATTACH 71
2.5.2.1 Parameters 72
2.5.2.2 Description 72
2.5.2.3 Example 72
2.6 Fabrication Process 72
2.6.1 DOPING 73
2.6.2 Analytic PROFILE 73
2.6.2.1 Parameters 74
2.6.2.2 Analytic Profile Calculation 77
2.6.2.3 Analytic2 Profile Calculation 78
2.6.2.4 Custom Doping Profile 78
2.6.2.5 Example 78
2.6.3 MOLE 79
2.6.3.1 Parameters 79
2.6.3.2 Description 80
2.6.3.3 Example 80
2.7 Physical Models 80
2.7.1 PMI 80
2.7.1.1 Parameters 81
2.7.1.2 Description 83
2.7.1.3 Mobility Models 84
2.7.1.4 Impact Ionization Models 84
2.7.1.5 Trap Model 85
2.7.1.6 Incomplete Ionization Model 86
2.7.2 MODEL 87
2.7.2.1 Parameters 88

2.7.2.2 Example 90

2.8 Numerical Solution 90

2.8.1 METHOD 90

2.8.1.1 Parameters 90

2.8.1.2 Nonlinear Solver Algorithms 95

2.8.1.3 Nonlinear Convergence Criteria 95

2.8.1.4 Linear Solver Algorithms 96

2.8.2 SOLVE 96

2.8.2.1 Parameters 97

2.8.2.2 Device DC Sweep Mode 102

2.8.2.3 Device Transient Mode 102

2.8.2.4 Device Operation Point Mode 103

2.8.2.5 Device Small-Signal AC Sweep Mode 104

2.8.2.6 Device/Circuit Mixed Operation Point Mode . 104

2.8.2.7 Device/Circuit Mixed DC Sweep Mode 104

2.8.2.8 Device/Circuit Mixed Transient Mode 104

2.8.2.9 Example 104

2.8.3 EMFEM2D 105

2.8.3.1 Parameters 105

2.8.3.2 Description 106

2.8.4 RAYTRACE 106

2.8.4.1 Parameters 107

2.8.4.2 Description 108

2.8.4.3 Example 109

2.9 Input and Output 109

2.9.1 IMPORT 109

2.9.1.1 Parameters 109
2.9.1.2 description 109
2.9.1.3 Example 110
2.9.2 EXPORT 110
2.9.2.1 Parameters 110
2.9.2.2 description 110
2.9.2.3 Example 111
2.9.3 HOOK 111
2.9.3.1 Parameters 111
2.9.3.2 Description 111
2.9.3.3 Default Hook Functions 112
2.9.3.4 CV Postprocessor 112
2.9.3.5 Probe Postprocessor 112
2.9.3.6 Visualization and Animation Postprocessor . . 113
2.9.3.7 Interface to Visit visualizer 113

3 Extending Genius Simulator 114

3.1 Custom Material Models 114
3.1.1 Overview of PMI 114
3.1.1.1 Files 114
3.1.1.2 Classes 116
3.1.1.3 How a PMI Library is Loaded 116
3.1.1.4 How a PMI Library is Used 117
3.1.2 Procedure of Building a Custom Material Library 119
3.1.2.1 Building the Example 119
3.1.2.2 Create a New Material Library 119

3.1.3 Class References 120
3.1.3.1 PMI_Server Class 120
3.1.3.2 PMIS_Server Class 121
3.1.3.3 PMIS_BasicParameter Class 121
3.1.3.4 PMIS_BandStructure Class 122
3.1.3.5 PMIS_Mobility 126
3.1.3.6 PMIS_Avalanche 127
3.1.3.7 PMIS_Trap 128
3.1.3.8 PMIS_Thermal 128
3.1.3.9 PMIS_Optical 128
3.1.3.10 PMII 128
3.1.3.11 PMII_BasicParameter 128
3.1.3.12 PMII_Thermal 129
3.1.3.13 PMII_Optical 129
3.1.3.14 PMIC 129
3.1.3.15 PMIC_BasicParameter 129
3.1.3.16 PMIC_Thermal 129
3.1.3.17 PMIC_Optical 129
3.2 Hooks 129
3.3 Python Interface 130
3.3.1 HTTP Frontend 130

CHAPTER 1

Physics in Genius Device Simulator

Since Gummel's original work, the drift-diffusion model has been widely used in the semiconductor device simulation. It is now the de facto industry standard in this field.

The original DD model can be achieved by following approximation from hydrodynamic model:

- Light speed is much faster than carrier speed.
- All the collision is elastic.
- Bandgap does not change during collision.
- Carrier temperature equals to lattice temperature and keeps equilibrium.
- The gradient of driving force should keep small.
- Carrier degenerate can be neglected.

Some improvements have been applied to DD model for extend its capability. These "patches" of course make things complex, but they can deal with real problems.

This chapter describes the DD model and its variations used by GENIUS code for describing semiconductor device behavior as well as physical based parameters such as mobility, recombination rate and son on.

Level 1 Drift-Diffusion Equation

Level 1 Drift-Diffusion (**DDML1**) is the fundamental solver of GENIUS code for lattice temperature keeps constant though out the solve procedure.

The primary function of **DDML1** is to solve the following set of partial differential equations, namely Poisson's equation, along with the hole and electron continuity equations:

Poisson's Equation

$$\nabla \cdot \varepsilon \nabla \psi = -q \left(p - n + {N_D}^{+} - {N_A}^{-} \right) \tag{1.1}$$

where, ψ is the electrostatic potential of the vacuum level. This choice makes the description of metal-oxide-semiconductor contact and heterojunction easier. n and p are the electron and hole concentration, ${N_D}^{+}$ and ${N_A}^{-}$ are the ionized impurity concentrations. q is the magnitude of the charge of an electron.

The relationship of conduct band E_c, valence band E_v and vacuum level ψ is:

$$E_c = -q\psi - \chi - \Delta E_c \tag{1.2a}$$
$$E_v = E_c - E_g + \Delta E_v. \tag{1.2b}$$

Here, χ is the electron affinity. E_g is the bandgap of semiconductor. ΔE_c and ΔE_v are the bandgap shift caused by heavy doping or mechanical strain.

Furthermore, the relationship of vacuum level ψ and intrinsic Fermi potential $\psi_{\text{intrinsic}}$ is:

$$\psi = \psi_{\text{intrinsic}} - \frac{\chi}{q} - \frac{E_g}{2q} - \frac{k_b T}{2q} \ln \left(\frac{N_c}{N_v} \right) \tag{1.3}$$

The reference 0eV of energy is set to intrinsic Fermi level of equilibrium state in GENIUS.

Continuity Equations

The continuity equations for electrons and holes are defined as follows:

$$\frac{\partial n}{\partial t} = \frac{1}{q} \nabla \cdot \vec{J}_n - (U - G) \tag{1.4a}$$
$$\frac{\partial p}{\partial t} = -\frac{1}{q} \nabla \cdot \vec{J}_p - (U - G) \tag{1.4b}$$

where $\vec{J}_n$ and $\vec{J}_p$ are the electron and hole current densities, U and G are the recombination and generation rates for both electrons and holes.

Drift-Diffusion Current Equations

The current densities $\vec{J}_n$ and $\vec{J}_p$ are expressed in terms of the level 1 drift-diffusion model here.

$$\vec{J}_n = q\mu_n n \vec{E}_n + qD_n \nabla n \tag{1.5a}$$
$$\vec{J}_p = q\mu_p p \vec{E}_p - qD_p \nabla p \tag{1.5b}$$

where μ_n and μ_p are the electron and hole mobilities. $D_n = \frac{k_b T}{q} \mu_n$ and $D_p = \frac{k_b T}{q} \mu_p$ are the electron and hole diffusivities, according to Einstein relationship.

Effective Electrical Field

$\vec{E}_n$ and $\vec{E}_p$ are the effective driving electrical field to electrons and holes, which related to local band diagram. The band structure of heterojunction has been taken into account here [, Lindefelt1994].

$$\vec{E}_n = \frac{1}{q}\nabla E_c - \frac{k_b T}{q}\nabla\left(\ln(N_c) - \ln(T^{3/2})\right) \tag{1.6a}$$
$$\vec{E}_p = \frac{1}{q}\nabla E_v + \frac{k_b T}{q}\nabla\left(\ln(N_v) - \ln(T^{3/2})\right) \tag{1.6b}$$

The lattice temperature keeps uniform throughout **DDML1**, the above temperature gradient item takes no effect in fact.

By substituting drift-diffusion model into the current density expressions, and combining with Poisson's equation, the following basic equations for **DDML1** are obtained:

$$\frac{\partial n}{\partial t} = \nabla\cdot\left(\mu_n n\vec{E}_n + \mu_n\frac{k_b T}{q}\nabla n\right) - (U-G) \tag{1.7a}$$
$$\frac{\partial p}{\partial t} = -\nabla\cdot\left(\mu_p p\vec{E}_p - \mu_p\frac{k_b T}{q}\nabla p\right) - (U-G) \tag{1.7b}$$
$$\nabla\cdot\varepsilon\nabla\psi = -q(p - n + {N_D}^+ - {N_A}^-) \tag{1.7c}$$

DDML1 is suitable for PN diode, BJT transistor and long gate MOSFET simulation. It is robust, and runs pretty fast for real work. The detailed discretization scheme can be found at [[TODO]].

Level 2 Drift-Diffusion Equation

The Level 2 DD model considers the influence of lattice temperature by solving the extra thermal equation simultaneously with the electrical equations. Also, the formula of drift-diffusion equation should be modified according to [, Selberherr1984].

The electron diffusion current in **DDML1** can be written as:

$$\vec{J}_{n,\mathrm{diff}} = \frac{k_b T}{q}\mu_n q\nabla n = k_b T\mu_n\nabla n \tag{1.8}$$

But for DDML2, it has the form of

Temperature Gradient Correction

$$\vec{J}_{n,\mathrm{diff}} = \mu_n k_b(T\nabla n + n\nabla T) \tag{1.9}$$

The hole diffusion current should be modified in the same manner.

$$\vec{J}_{p,\text{diff}} = -\mu_p k_b (T\nabla p + p\nabla T) \tag{1.10}$$

The following heat flow equation is used:

Heat Flow Equation

$$\rho c_p \frac{\partial T}{\partial t} = \nabla \cdot \kappa \nabla T + \vec{J} \cdot \vec{E} + (E_g + 3k_b T) \cdot (U - G) \tag{1.11}$$

where ρ is the mass density of semiconductor material. c_p is the heat capacity. κ is the thermal conductivity of the material. $\vec{J} \cdot \vec{E}$ is the joule heating of current. $(E_g + 3k_b T) \cdot (U - G)$ is lattice heating due to carrier recombination and generation.

From above discussion, the governing equations for DDML2 are as follows:

$$\frac{\partial n}{\partial t} = \nabla \cdot \left(\mu_n n \vec{E}_n + \mu_n \frac{k_b T}{q} \nabla n + \mu_n \frac{k_b \nabla T}{q} n\right) - (U - G) \tag{1.12a}$$

$$\frac{\partial p}{\partial t} = -\nabla \cdot \left(\mu_p p \vec{E}_p - \mu_p \frac{k_b T}{q} \nabla p - \mu_p \frac{k_b \nabla T}{q} p\right) - (U - G) \tag{1.12b}$$

$$\nabla \cdot \varepsilon \nabla \psi = -q\left(p - n + {N_D}^+ - {N_A}^-\right) \tag{1.12c}$$

$$\rho c_p \frac{\partial T}{\partial t} = \nabla \cdot \kappa \nabla T + \vec{J} \cdot \vec{E} + (E_g + 3k_b T) \cdot (U - G) \tag{1.12d}$$

This model can be used as power transistor simulation as well as breakdown simulation. Unfortunately, nearly all the physical parameters are related with temperature. They should be considered during self consistent simulation, which greatly slows down the speed. The $\mathbf{DDML2}$ solver runs $50 - 70\,\%$ slower than $\mathbf{DDML1}$. However, it seems no convergence degradation happens in most of the case. The discretization scheme can be found at [[TODO]].

Level 3 Energy Balance Equation

Energy Balance Model [, PISCES-2ET] is introduced into GENIUS code for simulating short channel MOSFET. This is a simplification of full hydrodynamic (HD) model [, Aste2003]. The current density expressions from the drift-diffusion model are modified to include additional coupling to the carrier temperature. Also, reduced carrier energy conservation equations, which derived from second order moment of Boltzmann Transport Equation, are solved consistently with drift-diffusion model. The simplification from HD to EB makes sophisticated Scharfetter-Gummel discretization still can be used in the numerical solution, which ensures the stability.

Current Equation for EBM

The current density $\vec{J}_n$ and $\vec{J}_p$ are then expressed as:

$$\vec{J}_n = q\mu_n n\vec{E}_n + k_b\mu_n\left(n\nabla T_n + T_n\nabla n\right) \tag{1.13a}$$

$$\vec{J}_p = q\mu_p p\vec{E}_p - k_b\mu_p\left(p\nabla T_p + T_p\nabla p\right) \tag{1.13b}$$

where, T_n and T_p are electron and hole temperature, respectively. The difference between above equations and carrier density equations in **DDML2** is lattice temperature replaced by carrier temperature.

Energy Balance Equations

In addition, the energy balance model includes the following electron and hole energy balance equations:

$$\frac{\partial\left(n\omega_n\right)}{\partial t} + \nabla\cdot\vec{S}_n = \vec{E}_n\cdot\vec{J}_n + H_n \tag{1.14a}$$

$$\frac{\partial\left(p\omega_p\right)}{\partial t} + \nabla\cdot\vec{S}_p = \vec{E}_p\cdot\vec{J}_p + H_p \tag{1.14b}$$

where, ω_n and ω_p are electron and hole energy. For HD model, the carrier energy includes thermal and kinetic terms $\omega_c = \frac{3}{2}k_bT_c + \frac{1}{2}m^*v_c^2$, but only thermal energy for EB model $\omega_c = \frac{3}{2}k_bT_c$. Here c stands for n or p. $\omega_0 = \frac{3}{2}k_bT$ is the carrier equilibrium energy, for carrier temperature equals to lattice temperature.

$\vec{S}_n$ and $\vec{S}_p$ are the flux of energy:

$$\vec{S}_n = -\kappa_n\nabla T_n - \left(\omega_n + k_bT_n\right)\frac{\vec{J}_n}{q} \tag{1.15a}$$

$$\vec{S}_p = -\kappa_p\nabla T_p + \left(\omega_p + k_bT_p\right)\frac{\vec{J}_p}{q} \tag{1.15b}$$

The heat conductivity parameter for carriers can be expressed as:

$$\kappa_c = (\frac{2}{5} + \gamma)\frac{{k_b}^2}{q}T_c\mu_c c \tag{1.16}$$

where c stands for n and p, respectively. The constant parameter γ equals -0.7 in the GENIUS software.

The H_n and H_p are the rate of net loss of carrier kinetic energy:

$$H_n = (R_{\text{Aug}} - G) \cdot \left(E_g + \frac{3k_bT_p}{2} \right) - \frac{3k_bT_n}{2} (R_{\text{SHR}} + R_{\text{dir}} - G) \tag{1.17a}$$

$$- \frac{n(\omega_n - \omega_0)}{\tau_n} \tag{1.17b}$$

$$H_p = (R_{\text{Aug}} - G) \cdot \left(E_g + \frac{3k_bT_n}{2} \right) - \frac{3k_bT_p}{2} (R_{\text{SHR}} + R_{\text{dir}} - G) \tag{1.17c}$$

$$- \frac{p(\omega_p - \omega_0)}{\tau_p} \tag{1.17d}$$

At last, the lattice heat flow equation should be rewritten as:

Lattice Heat Equation for EBM

$$\rho c_p \frac{\partial T}{\partial t} = \nabla \cdot \kappa \nabla T + H \tag{1.18}$$

where

$$H = R_{\text{SHR}} \cdot \left(E_g + \frac{3k_bT_p}{2} + \frac{3k_bT_n}{2} \right) + \frac{n(\omega_n - \omega_0)}{\tau_n} + \frac{p(\omega_p - \omega_0)}{\tau_p} \tag{1.19}$$

The carrier energy is mainly contributed by joule heating term $\vec{E}_c \cdot \vec{J}_c$, and heating (cooling) due to carrier generation (recombination) term. The carriers exchange energy with lattice by collision, which described by energy relaxation term τ_{ω_c}. This model is suitable for sub-micron MOS (channel length $1 \sim 0.1$ μm) and advanced BJT simulation. However, the computation burden of EB method is much higher than DD. And the convergence of EB solver is difficult to achieve, which requires more strict initial value and more powerful inner linear solver. The discretization scheme can be found at [[TODO]].

From above discussion, all the governing equations of DD/EB method is elliptical or parabolic. From mathematic point of view, does not like hyperbolic system[1], the solution of elliptical or parabolic system is always smooth. The required numerical technique is simple and mature for these systems. As a result, the DD and EB method is preferred against full hydrodynamic method.

Band Structure Model

The band structure parameters, including bandgap E_g, effective density of states in the conduction band N_c and valence band N_v, and intrinsic carrier concentration n_{ie}, are the most important and fundamental physical parameters for semiconductor material [, Sze1981].

Effective Density of States

[1]One have to face discontinuous problem, i.e. shock wave.

Effective density of states in the conduction and valence band are defined as follows:

$$N_c \equiv 2\left(\frac{m_n{}^* k_b T}{2\pi\hbar^2}\right)^{3/2} \tag{1.20a}$$

$$N_v \equiv 2\left(\frac{m_p{}^* k_b T}{2\pi\hbar^2}\right)^{3/2} \tag{1.20b}$$

The temperature dependencies of effective density of states is fairly simple:

$$N_c\left(T\right) = N_c\left(300\text{K}\right)\left(\frac{T}{300\text{K}}\right)^{1.5} \tag{1.21a}$$

$$N_v\left(T\right) = N_v\left(300\text{K}\right)\left(\frac{T}{300\text{K}}\right)^{1.5} \tag{1.21b}$$

The bandgap in GENIUS is expressed as follows:

Bandgap

$$E_g(T) = E_g(0) - \frac{\alpha T^2}{T+\beta} \tag{1.22a}$$

$$= E_g(300) + \alpha\left[\frac{300^2}{300+\beta} - \frac{T^2}{T+\beta}\right] \tag{1.22b}$$

When bandgap narrowing effects due to heavy doping takes place [, Slotboom1977], the band edge shifts:

Bandgap Narrowing due to Heavy Doping

$$\Delta E_g = \frac{E_{\text{bgn}}}{2k_b T}\left[\ln\frac{N_{\text{total}}}{N_{\text{ref}}} + \sqrt{\left(\ln\frac{N_{\text{total}}}{N_{\text{ref}}}\right)^2 + C_{\text{bgn}}}\right]. \tag{1.23}$$

The intrinsic concentration should be modified:

$$n_{ie} = \sqrt{N_c N_v}\exp\left(-\frac{E_g}{2k_b T}\right)\cdot\exp(\Delta E_g) \tag{1.24}$$

Since the carrier current equation (**??**), p. 12 involves the energy level of conduction band N_c and valence band N_v, the bandgap shift should be attributed to them. The bandgap narrowing is attributed half to the conduction band and another half to the valence band as default:

$$E_c' = E_c - \frac{1}{2}\Delta E_g \tag{1.25a}$$

$$E_v' = E_v + \frac{1}{2}\Delta E_g \tag{1.25b}$$

The parameters used in the default band structure model is listed in table **??**, p. 17.

Table 1.1: Parameters of the Default band structure model

Symbol	Parameter	Unit	Silicon	GaAs
$E_g(300)$	**EG300**	eV	1.1241	1.424
α	**EGALPH**	eV/K	2.73×10^{-4}	5.405×10^{-4}
β	**EGBETA**	K	0	204
E_{bgn}	**V0.BGN**	eV	6.92×10^{-3}	0
N_{ref}	**N0.BGN**	cm^3	1.30×10^{17}	1×10^{17}
C_{bgn}	**CON.BGN**	-	0.5	0.5
m_n	**ELECMASS**	m_0	1.0903	0.067
m_p	**HOLEMASS**	m_0	1.1525	0.6415
$N_c(300)$	**NC300**	cm^3	2.86×10^{19}	4.7×10^{17}
$N_v(300)$	**NV300**	cm^3	3.10×10^{19}	7.0×10^{18}

Band structure of compound semiconductors

[[TODO]]

Band Structure of SiGe

[[TODO]]

Band Structure of Tertiary Compound Semiconductor

[[TODO]]

Bandgap

[[TODO]]

Electron Affinity

[[TODO]]

Effective Mass

[[TODO]]

Density of States

[[TODO]]

Schenk's Bandgap Narrowing Model

[[TODO]] Equations of Schenk's model

The Schenk's bandgap narrowing model is available for silicon, and can be loaded with the option **Schenk** in the **PMI** command.

Carrier Recombination

Three recombination mechanisms are considered in GENIUS at present, including Shockley-Read-Hall, Auger, and direct (or radiative) recombination. The total recombination is considered as the sum of all:

$$U = U_n = U_p = U_{\mathrm{SRH}} + U_{\mathrm{dir}} + U_{\mathrm{Auger}} \tag{1.26}$$

where U_{SRH}, U_{dir} and U_{Auger} are SRH recombination, direct recombination and Auger recombination, respectively.

SRH Recombination

Shockley-Read-Hall (SRH) recombination see index "Shockley-Read-Hall recombination" rate is determined by the following formula:

$$U_{\mathrm{SRH}} = \frac{pn - {n_{ie}}^2}{\tau_p\left[n + n_{ie}\exp\left(\frac{\mathbf{ETRAP}}{kT_L}\right)\right] + \tau_n\left[p + n_{ie}\exp\left(\frac{-\mathbf{ETRAP}}{kT_L}\right)\right]} \tag{1.27}$$

where τ_n and τ_p are carrier life time, which dependent on impurity concentration [, Roulston1982].

$$\tau_n = \frac{\mathbf{TAUN0}}{1 + N_{\mathrm{total}}/\mathbf{NSRHN}} \tag{1.28a}$$

$$\tau_p = \frac{\mathbf{TAUP0}}{1 + N_{\mathrm{total}}/\mathbf{NSRHP}} \tag{1.28b}$$

Auger Recombination

The Auger recombination is a three-carrier recombination process, involving either two electrons and one hole or two holes and one electron. This mechanism becomes important when carrier concentration is large.

$$U_{\mathrm{Auger}} = \mathbf{AUGN}\left(pn^2 - n{n_{ie}}^2\right) + \mathbf{AUGP}(np^2 - p{n_{ie}}^2) \tag{1.29}$$

where **AUGN** and **AUGP** are Auger coefficient for electrons and holes. The value of Auger recombination U_{Auger} can be negative some times, which refers to Auger generation.

Direct Recombination

The direct recombination model expresses the recombination rate as a function of the carrier concentrations n and p, and the effective intrinsic density n_{ie}:

$$U_{\mathrm{dir}} = \mathbf{DIRECT}(np - {n_{ie}}^2) \tag{1.30}$$

The default value of the recombination parameters are listed in table **??**, p. 19:

Table 1.2: Default values of recombination parameters

Parameter	Unit	Silicon	GaAs	Ge
ETRAP	eV	0	0	0
DIRECT	cm^3s^{-1}	1.1e-14	7.2e-10	6.41e-14
AUGN	cm^6s^{-1}	1.1e-30	1e-30	1e-30
AUGP	cm^6s^{-1}	0.3e-30	1e-29	1e-30
TAUN0	s	1e-7	5e-9	1e-7
TAUP0	s	1e-7	3e-6	1e-7
NSRHN	cm^3	5e16	5e17	5e16
NSRHP	cm^3	5e16	5e17	5e16

Surface Recombination At semiconductor-insulator interfaces, additional SRH recombination can be introduced. The surface recombination rate has the unit $cm^{-2}s{-}1$, and is calculated with

$$U_{\text{Surf}} = \frac{pn - {n_{ie}}^2}{\frac{1}{\textbf{STAUN}}(n + n_{ie}) + \frac{1}{\textbf{STAUP}}(p + n_{ie})}. \tag{1.31}$$

The surface recombination velocities, **STAUN** and **STAUP**, have the unit of cm/s, and the default value of 0.

Mobility Models

Carrier mobility is one of the most important parameters in the carrier transport model. The DD model itself, developed at early 1980s, is still being used today due to advanced mobility model enlarged its ability to sub-micron device.

Mobility modeling is normally divided into: low field behavior, high field behavior and mobility in the (MOS) inversion layer.

The low electric field behavior has carriers almost in equilibrium with the lattice. The low-field mobility is commonly denoted by the symbol μ_{n0}, μ_{p0}. The value of this mobility is dependent upon phonon and impurity scattering. Both of which act to decrease the low field mobility. Since scattering mechanism is depended on lattice temperature, the low-field mobility is also a function of lattice temperature.

The high electric field behavior shows that the carrier mobility declines with electric field because the carriers that gain energy can take part in a wider range of scattering processes. The mean drift velocity no longer increases linearly with increasing electric field, but rises more slowly. Eventually, the velocity doesn't increase any more with increasing field but saturates at a constant velocity. This constant velocity is commonly denoted by the symbol v_{sat}. Impurity scattering is relatively insignificant for energetic carriers, and so v_{sat} is primarily a function of the lattice temperature.

Modeling carrier mobilities in inversion layers introduces additional complications. Carriers in inversion layers are subject to surface scattering, carrier-carrier scattering, velocity overshoot and quantum mechanical size quantization effects. These effects must be accounted for in order to perform accurate simulation of MOS devices. The transverse electric field is often used as a parameter that indicates the strength of inversion layer phenomena.

It can be seen that some physical mechanisms such as velocity overshoot and quantum effect which can't be described by DD method at all, can be taken into account by comprehensive mobility model. The comprehensive mobility model extends the application range of DD method. However, when the EB method (which accounts for velocity overshoot) and QDD method (including quantum effect) are used, more calibrations are needed to existing mobility models.

Bulk Mobility Models

The first family of mobility models were designed to model the carrier transport at low electric fields. They usually focus on the temperature and doping concentration dependence of the carrier mobilities. The surface-related or transverse E-field effects are *not* included in these models. On the other hand, in GENIUS, these low-field mobilities models are coupled to a velocity saturation model to account for the carrier velocity saturation effect. This family of mobility models are suitable for bulk device, such as bipolar transistors.

In brief, the low field carrier mobility is first computed, then a velocity saturation formula is applied to yield the corrected mobility value. Three choices are available for the low-field mobility calculation, each described in one of the following sub-sections. The choices of velocity saturation is described in the last sub-section.

Analytic Mobility Model

In the GENIUS code, the Analytic Mobility model [, Selberherr1984P] is the default low field mobility model for all the material. It is an concentration and temperature dependent empirical mobility model expressed as:

$$\mu_0 = \mu_{\text{min}} + \frac{\mu_{\text{max}} \left(\dfrac{T}{300}\right)^{\nu} - \mu_{\text{min}}}{1 + \left(\dfrac{T}{300}\right)^{\xi} \left(\dfrac{N_{\text{total}}}{N_{\text{ref}}}\right)^{\alpha}} \tag{1.32}$$

where $N_{\text{total}} = N_A + N_D$ is the total impurity concentration.

Default parameters for Si, GaAs and Ge are listed below:

Table 1.3: Default parameter values of the analytic mobility model

Symbol	Parameter	Unit	Si:n	Si:p	GaAs:n
μ_{min}	**MUN.MIN / MUP.MIN**	$cm^2V^{-1}s^{-1}$	55.24	49.70	0.0
μ_{max}	**MUN.MAX / MUP.MAX**	$cm^2V^{-1}s^{-1}$	1429.23	479.37	8500.0
ν	**NUN / NUP**	-	-2.3	-2.2	-1.0
ξ	**XIN / XIP**	-	-3.8	-3.7	0.0
α	**ALPHAN / ALPHAP**	-	0.73	0.70	0.436
N_{ref}	**NREFN / NREFP**	cm^{-3}	1.072e17	1.606e17	1.69e17

In GENIUS, the analytic model is the simplest mobility model, and is available for a wide range of materials. For some materials, such as silicon, some more advanced mobility models are available.

Masetti Analytic Model

The doping-dependent low-field mobility model proposed by Masetti et al.[, Masetti1983] is an alternative to the default analytic model. The general expression for the low-field mobility is

$$\mu_{\text{dop}} = \mu_{\text{min1}} \exp\left(-\frac{P_c}{N_{\text{tot}}}\right) + \frac{\mu_{\text{const}} - \mu_{\text{min2}}}{1 + (N_{\text{tot}}/C_r)^{\alpha}} - \frac{\mu_1}{1 + (C_s/N_{\text{tot}})^{\beta}} \tag{1.33}$$

where N_{tot} is the total doping concentration. The term μ_{const} is the temperature-dependent, phonon-limited mobility

$$\mu_{\text{const}} = \mu_{\text{max}} \left(\frac{T}{300}\right)^{\zeta} \tag{1.34}$$

where T is the lattice temperature.

The parameters of the Masetti model is listed in table **??**, p. 21. The Masetti model is the default mobility model for the 4H-SiC material.

Table 1.4: Parameters of the Masetti mobility model

Symbol	Parameter	Unit	4H-SiC:n	4H-SiC:p
μ_{max}	**MUN.MAX / MUP.MAX**	$cm^2V^{-1}s^{-1}$	947.0	124.0
ζ	**MUN.ZETA / MUP.ZETA**	-	1.962	1.424
μ_{min1}	**MUN.MIN1 / MUP.MIN1**	$cm^2V^{-1}s^{-1}$	0	15.9
μ_{min2}	**MUN.MIN2 / MUP.MIN2**	$cm^2V^{-1}s^{-1}$	0	15.9
μ_1	**MUN1 / MUP1**	$cm^2V^{-1}s^{-1}$	0	0

P_c	**PCN / PCP**	cm^{-3}	0	0
C_r	**CRN / CRP**	cm^{-3}	1.94×10^{17}	1.76×10^{19}
C_s	**CSN / CSP**	cm^{-3}	0	0
α	**MUN.ALPHA / MUP.ALPHA**	-	0.61	0.34
β	**MUN.BETA / MUP.BETA**	-	0	0

Philips Mobility Model

Another low field mobility model implemented into GENIUS is the Philips Unified Mobility model [, Klaassen1992-1],[, Klaassen1992-2]. This model takes into account the distinct acceptor and donor scattering, carrier-carrier scattering and carrier screening, which is recommended for bipolar devices simulation.

The electron mobility is described by the following expressions:

$$\mu_{0,n}{}^{-1} = \mu_{\text{Lattice},n}{}^{-1} + \mu_{D+A+p}{}^{-1} \tag{1.35}$$

where $\mu_{0,n}$ is the total low field electron mobilities, $\mu_{\text{Lattice},n}$ is the electron mobilities due to lattice scattering, μ_{D+A+p} is the electron and hole mobilities due to donor (D), acceptor (A), screening (P) and carrier-carrier scattering.

$$\mu_{\text{Lattice},n} = \mu_{\max}\left(\frac{T}{300}\right)^{-2.285} \tag{1.36}$$

$$\mu_{D+A+p} = \mu_{1,n}\left(\frac{N_{\text{sc},n}}{N_{\text{sc,eff},n}}\right)\left(\frac{N_{\text{ref}}}{N_{\text{sc},n}}\right)^{\alpha} + \mu_{2,n}\left(\frac{n+p}{N_{\text{sc,eff},n}}\right) \tag{1.37}$$

The parameters $\mu_{1,n}$ and $\mu_{2,n}$ are given as:

$$\mu_{1,n} = \frac{\mu_{\max}^2}{\mu_{\max} - \mu_{\min}}\left(\frac{T}{300}\right)^{3\alpha-1.5} \tag{1.38a}$$

$$\mu_{2,n} = \frac{\mu_{\max}\cdot\mu_{\min}}{\mu_{\max} - \mu_{\min}}\left(\frac{300}{T}\right)^{1.5} \tag{1.38b}$$

where $N_{\text{sc},n}$ and $N_{\text{sc,eff},n}$ is the impurity-carrier scattering concentration and effect impurity-carrier scattering concentration given by:

$$N_{\text{sc},n} = N_D{}^* + N_A^* + p \tag{1.39a}$$

$$N_{\text{sc,eff},n} = N_D{}^* + N_A{}^* G\left(P_n\right) + \frac{p}{F\left(P_n\right)} \tag{1.39b}$$

where N_D^* and N_A^* take ultra-high doping effects into account and are defined by:

$$N_D{}^* = N_D \left(1 + \frac{1}{C_D + \left(\frac{N_{D,\mathrm{ref}}}{N_D} \right)^2} \right) \tag{1.40a}$$

$$N_A{}^* = N_A \left(1 + \frac{1}{C_A + \left(\frac{N_{A,\mathrm{ref}}}{N_A} \right)^2} \right) \tag{1.40b}$$

The screening factor functions $G\left(P_n\right)$ and $F\left(P_n\right)$ take the repulsive potential for acceptors and the finite mass of scattering holes into account.

$$G\left(P_n\right) = 1 - \frac{0.89233}{\left[0.41372 + P_n \left(\frac{m_0}{m_e}\frac{T}{300}\right)^{0.28227}\right]^{0.19778}} + \frac{0.005978}{\left[P_n \left(\frac{m_e}{m_0}\frac{T}{300}\right)^{0.72169}\right]^{1.80618}} \tag{1.41}$$

$$F\left(P_n\right) = \frac{0.7643 P_n{}^{0.6478} + 2.2999 + 6.5502\frac{m_e}{m_h}}{P_n{}^{0.6478} + 2.3670 - 0.8552\frac{m_e}{m_h}} \tag{1.42}$$

The P_n parameter that takes screening effects into account is given by:

$$P_n = \left[\frac{f_{cw}}{N_{\mathrm{sc,ref}} \cdot N_{\mathrm{sc},n}{}^{-2/3}} + \frac{f_{BH}}{\frac{N_{\mathrm{c,ref}}}{n+p}\left(\frac{m_e}{m_0}\right)} \right]^{-1} \left(\frac{T}{300}\right)^2 \tag{1.43}$$

Similar expressions hold for holes. The default parameters for Philips model are listed in table **??**, p. 23:

Table 1.5: Default values of Philips mobility model parameters

Symbol	Parameter	Unit	Si:n	Si:p
μ_{min}	**MMNN.UM / MMNP.UM**	$cm^2V^{-1}s^{-1}$	55.24	49.70
μ_{max}	**MMXN.UM / MMXP.UM**	$cm^2V^{-1}s^{-1}$	1417.0	470.5
α	**ALPN.UM / ALPP.UM**	-	0.68	0.719
N_{ref}	**NRFN.UM / NRFP.UM**	cm^{-3}	9.68e16	2.23e17
C_D	**CRFD.UM**	-	0.21	0.21
C_A	**CRFA.UM**	-	0.5	0.5
$N_{\mathrm{D,ref}}$	**NRFD.UM**	cm^{-3}	4.0e20	4.0e20
$N_{\mathrm{A,ref}}$	**NRFA.UM**	cm^{-3}	7.2e20	7.2e20
m_e	$\mathbf{me_over_m0}$	m_0	1.0	-
m_h	$\mathbf{mh_over_m0}$	m_0	-	1.258

f_{cw}		-	2.459	2.459
f_{BH}		-	3.828	3.828
$N_{\mathrm{sc,ref}}$	**NSC.REF**	cm^{-2}	3.97e13	3.97e13
$N_{\mathrm{c,ref}}$	**CAR.REF**	cm^{-3}	1.36e20	1.36e20

In the actual code, Philips model is corrected by Caughey-Thomas expression for taking high field velocity saturation effects into account. This model can be loaded by **Philips** keyword in the **PMI** statements.

Velocity Saturation

Silicon-like materials For silicon-like materials, the Caughey-Thomas expression [, Caughey1967], is used:

$$\mu = \frac{\mu_0}{\left[1 + \left(\frac{\mu_0 E_{\parallel}}{v_{\mathrm{sat}}}\right)^{\beta}\right]^{1/\beta}} \tag{1.44}$$

where $E_{\parallel}$ is the electric field parallel to current flow. v_{sat} is the saturation velocities for electrons or holes. They are computed by default from the expression:

$$v_{\mathrm{sat}}(T) = \frac{v_{\mathrm{sat0}}}{1 + \alpha \cdot \exp\left(\frac{T}{600}\right)} \tag{1.45}$$

The parameters and the default values for silicon is listed in table **??**, p. 24.

Table 1.6: Velocity saturation parameters of silicon-like materials

Symbol	Parameter	Unit	Si:n	Si:p
v_{sat0}	**VSATN0 / VSATP0**	cm/s	2.4×10^7	2.4×10^7
β	**BETAN / BETAP**	-	2.0	1.0
α	**VSATN.A / VSATP.A**	-	0.8	0.8

GaAs-like materials For GaAs-like materials, another expression due to [, Barnes1976] is used to describe the negative differential resistance:

$$u = \frac{\mu_0 + \frac{v_{sat}}{E_{\parallel}}\left(\frac{E_{\parallel}}{E_0}\right)^4}{1 + \left(\frac{E_{\parallel}}{E_0}\right)^4} \tag{1.46}$$

where E_0 is the reference field, and the saturation velocity

$$v_{sat}(T) = A_{\mathrm{vsat}} - B_{\mathrm{vsat}}T \tag{1.47}$$

The negative differential property of carrier mobility is described in this model. When electric field increases in this model, the carrier drift velocity ($\mu E_{\|}$) reaches a peak and then begins to decrease at high fields due to the transferred electron effect.

The parameters are listed in table **??**, p. 25.

Table 1.7: Velocity saturation parameters of GaAs-like materials

Symbol	Parameter	Unit	GaAs:n	GaAs:p
A_{vsat}	**VSATN.A / VSATP.B**	cm/s	1.13×10^7	1.13×10^7
B_{vsat}	**VSATN.A / VSATP.B**	cm/s/K	1.2×10^4	1.2×10^4

GaAs-specific model When using this model for GaAs MESFET device simulation, the negative differential property may cause the drain output characteristics (current vs. voltage) exhibit an unrealistic oscillation behavior. Another model to describe high field effects developed by Yeager [, Yeager1986] can be used.

$$\mu = \frac{v_{\mathrm{sat}}}{E_{\|}} \tanh\left(\frac{\mu_0 E_{\|}}{v_{\mathrm{sat}}}\right) \tag{1.48}$$

This GaAs-specific model can be loaded by **Hypertang** keyword in **PMI** statement.

4H-SiC-specific model For 4H-SiC, the saturation velocity is calculated with the following formula

$$v_{sat}(T) = A_{\mathrm{vsat}} - B_{\mathrm{vsat}}\left(\frac{T}{300}\right) \tag{1.49}$$

where the parameters are listed in

Table 1.8: Velocity saturation parameters of 4H-SiC

Symbol	Parameter	Unit	4H-SiC:n	4H-SiC:p
A_{vsat}	**VSATN.A / VSATP.B**	cm/s	1.07×10^7	8.37×10^6
B_{vsat}	**VSATN.A / VSATP.B**	cm/s	0	0

Unified Mobility Models

The other family of mobility models are the unified mobility models. The effect of high transverse and parallel E-field is an integral part in the design of these mobility models. As a result, these models are recommended for silicon MOSFET simulation. On the other hand, the availability of the unified models is limited to a few materials, such as silicon and silicon-germanium.

Lombardi Surface Mobility Model

Along an insulator-semiconductor interface, the carrier mobilities can be substantially lower than in the bulk of the semiconductor due to surface-related scattering. If no surface degradation is considered, the drain-source current may exceed about $30\,\%$ for MOS simulation.

The Lombardi mobility model [, Lombardi1988] is an empirical model that is able to describe the carrier mobility in the MOSFET inversion layer. The Lombardi model consists of three components

- μ_b, the doping-dependent bulk mobility. This component mainly accounts for the ionized impurity scattering.
- μ_{ac}, the mobility degradation due to acoustic phonon scattering in the inversion layer. Due to the quantum confinement in the potential well at the interface, this mobility degradation is a strong function of the transverse electric field.
- μ_{sr}, the mobility degradation due to the surface roughness scattering. This component is also a strong function of the transverse electric field.

To obtain the final value of carrier mobility, the three components are combined using the Matthiessen's rule:

$$\mu_{\mathrm{s}}{}^{-1} = \mu_b{}^{-1} + \mu_{\mathrm{ac}}{}^{-1} + \mu_{\mathrm{sr}}{}^{-1}. \tag{1.50}$$

Bulk Component

The bulk mobility component in Lombardi's model is similar to that of Masetti's model, which reads

$$\mu_b = \mu_0 \exp\left(-\frac{P_c}{N_{\mathrm{tot}}}\right) + \frac{\mu_{\max} - \mu_0}{1 + (N_{\mathrm{tot}}/C_r)^{\alpha}} - \frac{\mu_1}{1 + (C_s/N_{\mathrm{tot}})^{\beta}} \tag{1.51}$$

$$\mu_{\max} = \mu_2 \left(\frac{T}{300}\right)^{\zeta} \tag{1.52}$$

Acoustic Phonon Component

The acoustic phonon limited mobility component is

$$\mu_{\mathrm{ac}} = \frac{B}{E_\perp} + \frac{C \cdot N_{\mathrm{total}}{}^{\lambda}}{T \sqrt[3]{E_\perp}} \tag{1.53}$$

where $E_\perp$ is the transverse component of the electric field.

Surface Roughness Component

Finally, the surface roughness limited mobility is expressed as

$$\mu_{\mathrm{sr}} = \frac{D}{E_\perp^{\gamma}}. \tag{1.54}$$

The Lombardi model is uses the Caughey-Thomas model for velocity saturation calculation, see Velocity Saturation, p. 24 for details.

The parameters used in the Lombardi model is summarized in table **??**, p. 27.

Table 1.9: Parameters of Lombardi mobility model

Symbol	Parameter	Unit	Si:n	Si:p
α	**EXN1.LSM / EXP1.LSM**	-	0.680	0.719
β	**EXN2.LSM / EXP2.LSM**	-	2.0	2.0
ζ	**EXN3.LSM / EXP3.LSM**	-	2.5	2.2
λ	**EXN4.LSM / EXP4.LSM**	-	0.125	0.0317
γ	**EXN8.LSM / EXP8.LSM**	-	2.0	2.0
μ_0	**MUN0.LSM / MUP0.LSM**	$cm^2V^{-1}s^{-1}$	52.2	44.9
μ_1	**MUN1.LSM / MUP1.LSM**	$cm^2V^{-1}s^{-1}$	43.4	29.0
μ_2	**MUN2.LSM / MUP2.LSM**	$cm^2V^{-1}s^{-1}$	1417.0	470.5
P_c	**PC.LSM**	cm^{-3}	0 (fixed)	9.23×10^{16}
C_r	**CRN.LSM / CRP.LSM**	cm^{-3}	9.68×10^{16}	2.23×10^{17}
C_s	**CSN.LSM / CSP.LSM**	cm^{-3}	3.43×10^{20}	6.10×10^{20}
B	**BN.LSM / BP.LSM**	cm/s	4.75×10^7	9.93×10^6
C	**CN.LSM / CP.LSM**	-	1.74×10^5	8.84×10^5
D	**DN.LSM / DP.LSM**	-	5.82×10^{14}	2.05×10^{14}
v_{sat0}	**VSATN0 / VSATP0**	cm/s	2.4×10^7	2.4×10^7
β	**BETAN / BETAP**	-	2.0	1.0
α	**VSATN.A / VSATP.A**	-	0.8	0.8

Hewlett-Packard mobility model can be loaded by **Lombardi** keyword in the **PMI** statement.

Lucent High Field Mobility Model

The Lucent Mobility model [, Darwish1997] is an all-inclusive model which suitable for MOS simulation. This model incorporates Philips Unified Mobility model and the Lombardi Surface Mobility model, as well as accounting for high field effects. For low longitudinal field, the carrier mobility is given by Matthiessen's rule:

$$\mu_0 = \left[\frac{1}{\mu_{\text{b}}} + \frac{1}{\mu_{\text{ac}}} + \frac{1}{\mu_{\text{sr}}}\right]^{-1} \tag{1.55}$$

where μ_b is the bulk mobility comes from the Philips model, and μ_{ac} and μ_{sr} come from the Lombardi model. The details of these models are described in Philips Mobility Model, p. 22 and Lombardi Surface Mobility Model, p. 26, and will not be repeated here.

There is however a modification to the surface roughness formula equation (**??**),

p. 26. The constant exponent λ is replaced by the following function

$$\lambda = A + \frac{F \cdot (n+p)}{N_{\mathrm{tot}}^{\nu}}. \tag{1.56}$$

The Lucent model uses the Caughey-Thomas model for velocity saturation calculation, see Velocity Saturation, p. 24 for details.

The parameters of the Lucent model is listed in table **??**, p. 28.

aaaa Default values of Lucent mobility modelDefault values of Lucent mobility modelDefault values of Lucent mobility modelDefault values of Lucent mobility modelDefault values of Lucent mobility modelDefault values of Lucent mobility modelDefault values of Lucent mobility modelDefault values of Lucent mobility modelDefault values of Lucent mobility model

Table 1.10: Default values of Lucent mobility model parameters

Symbol	Parameter	Unit	Si:n	Si:p
μ_{min}	**MMNN.UM / MMNP.UM**	$cm^2V^{-1}s^{-1}$	55.24	49.70
μ_{max}	**MMXN.UM / MMXP.UM**	$cm^2V^{-1}s^{-1}$	1417.0	470.5
α	**ALPN.UM / ALPP.UM**	-	0.68	0.719
N_{ref}	**NRFN.UM / NRFP.UM**	cm^{-3}	9.68e16	2.23e17
C_D	**CRFD.UM**	-	0.21	0.21
C_A	**CRFA.UM**	-	0.5	0.5
$N_{\mathrm{D,ref}}$	**NRFD.UM**	cm^{-3}	4.0e20	4.0e20
$N_{\mathrm{A,ref}}$	**NRFA.UM**	cm^{-3}	7.2e20	7.2e20
m_e	$\mathbf{me_over_m0}$	m_0	1.0	-
m_h	$\mathbf{mh_over_m0}$	m_0	-	1.258
f_{cw}		-	2.459	2.459
f_{BH}		-	3.828	3.828
$N_{\mathrm{sc,ref}}$	**NSC.REF**	cm^{-2}	3.97e13	3.97e13
$N_{\mathrm{c,ref}}$	**CAR.REF**	cm^{-3}	1.36e20	1.36e20
v_{sat0}	**VSATN0 / VSATP0**	cm/s	2.4×10^7	2.47
λ	**EXN4.LUC / EXP4.LUC**	-	0.0233	0.0119
ν	**EXN9.LUC / EXP9.LUC**	-	0.0767	0.123
A	**AN.LUC / AP.LUC**	-	2.58	2.18
B	**BN.LUC / BP.LUC**	cm/s	3.61×10^7	1.51×10^7
C	**CN.LUC / CP.LUC**	-	1.70×10^4	4.18×10^3
D	**DN.LUC / DP.LUC**	-	3.58×10^{18}	3.58×10^{18}
F	**FN.LUC / FP.LUC**	-	6.85×10^{-21}	7.82×10^{-21}
K	**KN.LUC / KP.LUC**	-	1.70	0.90
v_{sat0}	**VSATN0 / VSATP0**	cm/s	2.4×10^7	2.4×10^7
β	**BETAN / BETAP**	-	2.0	1.0
α	**VSATN.A / VSATP.A**	-	0.8	0.8

The Lucent model can be loaded by **Lucent** keyword in the **PMI** statements. This is an accurate model recommended for MOS devices. However, its computation cost is higher than other mobility models. At the same time, it is also less numerically stable.

Hewlett-Packard High Field Mobility Model

It is reported that Hewlett-Packard mobility model [, Cham1986] achieves the same accuracy as Lucent model with relatively small computational burden in the MOS simulation.

This model also takes into account dependence on electric fields both parallel ($E_{\parallel}$) and perpendicular ($E_{\perp}$) to the direction of current flow. The mobility is calculated from

$$\mu = \frac{\mu_{\perp}}{\sqrt{1 + \frac{\left(\frac{\mu_{\perp}E_{\parallel}}{v_c}\right)^2}{\frac{\mu_{\perp}E_{\parallel}}{v_c} + \gamma} + \frac{\mu_{\perp}E_{\parallel}}{v_s}}} \tag{1.57}$$

where the tranverse field dependent component $\mu_{\perp}$ is given as:

$$\mu_{\perp} = \mu_0 \text{if } N_{\text{total}} > N_{\text{ref}} \frac{\mu_0}{1 + \frac{E_{\perp}}{E_{\text{ref}}}} \tag{1.58}$$

The low field mobility μ_0 is calculated from the **Analytic** model, as described in Analytic Mobility Model, p. 20.

The parameters of the Hewlett-Packard mobility model and its default values for silicon is listed in table **??**, p. 29.

Table 1.11: Default values of HP mobility model parameters

Symbol	Parameter	Unit	Si:n	Si:p
μ_{min}	**MUN.MIN / MUP.MIN**	$cm^2V^{-1}s^{-1}$	55.24	49.70
μ_{max}	**MUN.MAX / MUP.MAX**	$cm^2V^{-1}s^{-1}$	1429.23	479.37
ν	**NUN / NUP**	-	-2.3	-2.2
ξ	**XIN / XIP**	-	-3.8	-3.7
α	**ALPHAN / ALPHAP**	-	0.73	0.70
N_{ref}	**NREFN / NREFP**	cm^{-3}	1.072×10^{17}	1.606×10^{17}
μ_0	**MUN0.HP / MUP0.HP**	$cm^2V^{-1}s^{-1}$	774.0	250
v_c	**VCN.NP / VCP.HP**	$cm.s^{-1}$	4.9×10^6	2.928×10^6
v_s	**VSN.NP / VSP.HP**	$cm.s^{-1}$	1.036×10^7	1.2×10^7
γ	**GN.HP / GP.HP**	-	8.8	1.6
N_{ref}	**NREFN / NREFP**	cm^{-3}	5×10^{17}	5×10^{17}

E_{ref}	**ECN.HP / ECP.HP**	$V.cm^{-1}$	5.5×10^5	2.78×10^5

The Hewlett-Packard mobility model can be loaded by **HP** keyword in the **PMI** statement.

Mobility Models of Complex Compound Semiconductors

[[TODO]]

Carrier Temperature Based Mobility Model

We should notice here, all the above mobility models are developed under the framework of DD method. Since DD is an approximate model for semiconductor, the difference between DD model and real device is corrected by mobility models! These mobility model contains some physical model that DD does not consider. For example, the high field correction has already contains the effect of hot carriers. The surface mobility for MOSFET not only considers the mobility degrade due to surface roughness, but also contains the effect caused by carrier concentration decrease due to quantum well in inverse layer. These corrections extended the application range of DD model, also make the mobility model rather complex.

When the physical model is more accurate, the carrier mobility model can be less complicated. Thus, the mobility models suitable for DD model may NOT be suitable for energy balance model. There are some mobility models developed special for energy balance model [, PISCES-2ET]. However, they are yet to be implemented in GENIUS.

Generation Model

Impact Ionization

The generation rate of electron-hole pairs due to the carrier impact ionization (II) is generally modeled as [, Sze1981]:

$$G^{II} = \alpha_n \frac{\left|\vec{J}_n\right|}{q} + \alpha_p \frac{\left|\vec{J}_p\right|}{q} \tag{1.59}$$

where α_n and α_p are electron and hole ionization coefficients, related with electrical field, material and temperature.

Three models are implemented in Genius to calculate the ionization coefficient.

Selberherr Model

Selberherr gives an empirical formula [, Selberherr1984], based on the expression derived by Chynoweth[, Chynoweth1958]:

$$\alpha_{n,p} = \alpha_{n,p}^{\infty}(T) \exp\left[-\left(\frac{E_{n,p}^{\mathrm{Crit}}}{E_{n,p}}\right)^{\gamma_{n,p}}\right] \tag{1.60}$$

where $E_{n,p}$ is the magnitude of driving fields. When $\mathbf{EdotJ}$ model is used, $E_{n,p}$ can be given by:

$$E_n = \frac{\vec{E} \cdot \vec{J}_n}{\left|\vec{J}_n\right|} \tag{1.61a}$$

$$E_p = \frac{\vec{E} \cdot \vec{J}_p}{\left|\vec{J}_p\right|} \tag{1.61b}$$

and for $\mathbf{GradQf}$ model:

$$E_n = \left|\nabla \phi_{F_n}\right| \tag{1.62a}$$

$$E_p = \left|\nabla \phi_{F_p}\right| \tag{1.62b}$$

where $E_{n,p}^{\mathrm{Crit}} = \dfrac{E_g}{q\lambda_{n,p}}$, for which $\lambda_{n,p}$ are the optical-phonon mean free paths for electrons and holes given by:

$$\lambda_n(T) = \lambda_{n,0} \cdot \tanh\left(\frac{E_{\mathrm{op}}}{2k_bT}\right) \tag{1.63a}$$

$$\lambda_p(T) = \lambda_{p,0} \cdot \tanh\left(\frac{E_{\mathrm{op}}}{2k_bT}\right) \tag{1.63b}$$

in the above expressions, E_{op} is the optical-phonon energy. $\lambda_{n,0}$ and $\lambda_{p,0}$ are the phonon mean free paths for electrons and holes at 300K.

The temperature dependent factors α_n^{∞} and α_p^{∞} are expressed as:

$$\alpha_n^{\infty} = \alpha_{n,0} + \alpha_{n,1} \cdot T + \alpha_{n,2} \cdot T^2 \tag{1.64a}$$

$$\alpha_p^{\infty} = \alpha_{p,0} + \alpha_{p,1} \cdot T + \alpha_{p,2} \cdot T^2 \tag{1.64b}$$

The Selberherr model is the default avalanche model for many materials in Genius. One can also explicitly load it with the option $\mathbf{Selberherr}$ in the $\mathbf{PMI}$ statements.The parameters used for Selberherr model are listed in table **??**, p. 32.

Table 1.12: Default values of Impact Ionization model parameters

Symbol	Parameter	Unit	Silicon	GaAs	Ge
$\lambda_{n,0}$	**LAN300**	cm	1.04542e-6	3.52724e-6	6.88825e-7
$\lambda_{p,0}$	**LAP300**	cm	6.32079e-7	3.67649e-6	8.39505e-7
γ_n	**EXN.II**	-	1.0	1.6	1.0
γ_p	**EXP.II**	-	1.0	1.75	1.0
E_{op}	**OP.PH.EN**	eV	6.3e-2	3.5e-2	3.7e-2
$\alpha_{n,0}$	**N.IONIZA**	cm	7.030e5	2.994e5	1.55e7
$\alpha_{n,1}$	**N.ION.1**	cm	0.0	0.0	0.0
$\alpha_{n,2}$	**N.ION.2**	cm	0.0	0.0	0.0
$\alpha_{p,0}$	**P.IONIZA**	cm	1.528e6	2.215e5	1e7
$\alpha_{p,1}$	**P.ION.1**	cm	0.0	0.0	0.0
$\alpha_{p,2}$	**P.ION.2**	cm	0.0	0.0	0.0

van Overstraeten - de Man model

The model also uses the ionization coefficient derived by Chynoweth [, Chynoweth1958]

$$\alpha = \gamma a \exp\left(-\frac{\gamma b}{E}\right), \tag{1.65}$$

where

$$\gamma = \frac{\tanh\left(\frac{\hbar\omega_{\text{op}}}{2kT_0}\right)}{\tanh\left(\frac{\hbar\omega_{\text{op}}}{2kT}\right)}. \tag{1.66}$$

The van Overstraeten - de Man model uses two sets of values for parameter a and b at high and low electric field. The threshold for the switch is E_0.

The parameters are listed in table **??**, p. 32.

The van Overstraeten - de Man model is can be selected with the **vanOverstraetendeMan** option in the **PMI** command. It is the default model for 4H-SiC, InN, InAs and InSb materials.

Table 1.13: van Overstraeten - de Man Impact Ionization model parameters

Parameter	Parameter	Unit	4H-SiC:n	4H-SiC:p
b	**AN.II.LO / AP.II.LO**	cm	4.2×10^5	4.2×10^5
b	**AN.II.HI / AP.II.HI**	cm	4.2×10^5	4.2×10^5
b	**BN.II.LO / BP.II.LO**	V/cm	1.67×10^7	1.67×10^7
b	**BN.II.HI / BP.II.HI**	V/cm	1.67×10^7	1.67×10^7

E_0	**E0N.II / EOP.II**	V/cm	4×10^5	4×10^5
$\hbar\omega_{\rm op}$	**EN.OP.PH / EP.OP.PH**	eV	1.0	1.0

Valdinoci Model

GENIUS has another Valdinoci model for silicon device which has been reported to produce correct temperature dependence of breakdown voltage of junction diodes as high as $400°C$ [, Valdinoci1999]. It can be loaded with the **Valdinoci** option in the **PMI** statements.

The electron impact ionization rate for Valdinoci model reads:

$$\alpha_n = \frac{E_\parallel}{a_n(T) + b_n(T)\exp\left(\dfrac{d_n(T)}{E_\parallel + c_n(T)}\right)} \tag{1.67}$$

where

$$a_n(T) = \mathbf{A0N} + \mathbf{A1N} \cdot T^{\mathbf{A2N}} \tag{1.68a}$$
$$b_n(T) = \mathbf{B0N} \cdot \exp\left(\mathbf{B1N} \cdot T\right) \tag{1.68b}$$
$$c_n(T) = \mathbf{C0N} + \mathbf{C1N} \cdot T^{\mathbf{C2N}} + \mathbf{C3N} \cdot T^2 \tag{1.68c}$$
$$d_n(T) = \mathbf{D0N} + \mathbf{D1N} \cdot T + \mathbf{D2N} \cdot T^2 \tag{1.68d}$$

Similar expressions hold for holes. The parameters for Valdinoci model are listed in table **??**, p. 33.

Table 1.14: Default values of Valdinoci Impact Ionization model parameters

Parameter	Silicon:n	Parameter	Silicon:p	Unit
A0N	4.3383	**A0P**	2.376	V
A1N	-2.42×10^{-12}	**A1P**	0.01033	$V \times K^{A2X}$
A2N	4.1233	**A2P**	1.0	-
B0N	0.235	**B0P**	0.17714	V
B1N	0.0	**B1P**	-2.178×10^{-3}	K
C0N	1.6831×10^4	**C0P**	0.0	$V.cm^{-1}$
C1N	4.3796	**C1P**	9.47×10^{-3}	$V.cm^{-1}\mathrm{K}^{-\mathbf{C2X}}$
C2N	1.0	**C2P**	2.4924	-
C3N	0.13005	**C3P**	0.0	$V.cm^{-1}\mathrm{K}^{-2}$
D0N	1.233735×10^6	**D0P**	1.4043×10^6	$V.cm^{-1}$
D1N	1.2039×10^3	**D1P**	2.9744×10^3	$V.cm^{-1}$ K
D2N	0.56703	**D2P**	1.4829	$V.cm^{-1}\mathrm{K}^{-2}$

Band-to-band Tunneling

Kane's model The carrier generation by band-band tunneling G^{BB} is also considered by Genius, which can be expressed as [, Kane1959][, Liou1990]:

$$G^{BB} = \mathrm{A.BTBT} \cdot \frac{E^2}{\sqrt{E_g}} \cdot \exp\left(-\mathrm{B.BTBT} \cdot \frac{{E_g}^{3/2}}{E}\right) \tag{1.69}$$

where E is the magnitude of electrical field.

High Energy Particles

As a heavy ion passes through the device, it will interact with some silicon atoms and transfer energy to the semiconductor lattice, which generates electron-hole pairs along it trajectory. The simulation of high-energy particle and the energy deposition in semiconductor can be simulated with the Geant4 or other Monte Carlo simulation tool. On the other hand, the generation of electron-hole pairs and the effects to the device behavior must be simulated in a 3D device simulator.

Assuming the proton hit the diode at $t = 0$s, and the electron-hole generation rate follows a Gaussian time dependence, with a maximum at $t_{\max}$, and a characteristic time τ, the carrier generation rate can be calculated by

$$G = G_0 \exp\left[-\frac{(t - t_{\max})^2}{2\tau^2}\right]. \tag{1.70}$$

On the other hand, the Monte Carlo simulators such as Geant4 provides the total energy deposition data, which relate to G by

$$E = \eta \int\limits_0^\infty G_0 \exp\left[-\frac{(t - t_{\max})^2}{2\tau^2}\right] dt. \tag{1.71}$$

where η is the average energy loss for each generated electron-hole pair. We therefore have the normalization factor

$$G_0 = \frac{2E}{\eta\tau\sqrt{\pi}}. \tag{1.72}$$

Fermi-Dirac Statistics

In general, the electron and hole concentrations in semiconductors are defined by Fermi-Dirac distributions and density of states:

$$n = N_c \mathcal{F}_{1/2}\left(\eta_n\right) \tag{1.73a}$$
$$p = N_v \mathcal{F}_{1/2}\left(\eta_p\right) \tag{1.73b}$$

The η_n and η_p are defined as follows:

$$\eta_n = \frac{E_{F_n} - E_c}{k_b T} = {\mathcal{F}_{1/2}}^{-1}\left(\frac{n}{N_c}\right) \tag{1.74a}$$
$$\eta_p = \frac{E_v - E_{F_p}}{k_b T} = {\mathcal{F}_{1/2}}^{-1}\left(\frac{p}{N_v}\right) \tag{1.74b}$$

where E_{F_n} and E_{F_p} are the electron and hole Fermi energies. The relationship of Fermi energy and Fermi potential is $E_{F_n} = -q\phi_n$, $E_{F_p} = -q\phi_p$.

Evaluate Inverse Fermi Integral

${\mathcal{F}_{1/2}}^{-1}$ is the inverse Fermi integral of order one-half. Joyce and Dixon gives its approximation analytic expression in the year of 1977 [, Joyce1977], which can be given by:

$${\mathcal{F}_{1/2}}^{-1}(x) = \log(x) + ax + bx^2 + cx^3 + dx^4 x < 8.463 \left[\left(\frac{3\sqrt{\pi}}{4}x\right)^{3/4} - \frac{\pi^2}{6}\right]^{1/2} \text{otherwise} \tag{1.75}$$

$$a = 0.35355339059327379 \tag{1.76a}$$
$$b = 0.0049500897298752622 \tag{1.76b}$$
$$c = 1.4838577128872821 \times 10^{-4} \tag{1.76c}$$
$$d = 4.4256301190009895 \times 10^{-6} \tag{1.76d}$$

In the GENIUS code, the η_n and η_p are derived from carrier concentration by Joyce-Dixon expression.

For convenience, we introduce floowing two parameters as referred by [, SEDAN1985]:

$$\gamma_n = \frac{\mathcal{F}_{1/2}\left(\eta_n\right)}{\exp\left(\eta_n\right)} \tag{1.77a}$$
$$\gamma_p = \frac{\mathcal{F}_{1/2}\left(\eta_p\right)}{\exp\left(\eta_p\right)} \tag{1.77b}$$

The carrier concentration for Fermi statistics and Boltzmann statistics can be described uniformly by:

$$n = N_c \gamma_n \exp\left(\eta_n\right) \tag{1.78a}$$
$$p = N_v \gamma_p \exp\left(\eta_p\right) \tag{1.78b}$$

where $\gamma_n = \gamma_p = 1$ for Boltzmann statistics, and less than 1.0 for Fermi statistics.

DD Equation with Fermi Statistics

Consider the drift-diffusion current equation (**??**), p. 11 when the carrier satisfies Fermi statistics and forces zero net current in equilibrium state, one can get the modified current equation, for which the Einstein relationship:

$$D_n = \frac{k_b T}{q} \mu_n \tag{1.79a}$$
$$D_p = \frac{k_b T}{q} \mu_p \tag{1.79b}$$

should be replaced by:

$$D_n = \frac{k_b T}{q} \mu_n \mathcal{F}_{1/2}\left(\eta_n\right) / \mathcal{F}_{-1/2}\left(\eta_n\right) \tag{1.80a}$$
$$D_p = \frac{k_b T}{q} \mu_p \mathcal{F}_{1/2}\left(\eta_p\right) / \mathcal{F}_{-1/2}\left(\eta_p\right) \tag{1.80b}$$

where $\mathcal{F}_{-1/2}$ is the Fermi integral of order minus one-half. The corresponding current equation for electrons is

$$\vec{J}_n = \mu_n \left(q n \vec{E}_n + k_b T \lambda_n \nabla n \right) \tag{1.81}$$

where

$$\lambda_n = \frac{\mathcal{F}_{1/2}\left(\eta_n\right)}{\mathcal{F}_{-1/2}\left(\eta_n\right)} \tag{1.82}$$

The Fermi integral has an useful property:

$$\frac{\partial}{\partial \eta} \mathcal{F}_\nu(\eta) = \mathcal{F}_{\nu-1}(\eta) \tag{1.83}$$

From the above property, one can derive two useful derivatives:

$$\frac{\partial}{\partial n}\eta_n(n) = \frac{\lambda_n}{n} \tag{1.84a}$$

$$\frac{\partial}{\partial n}\gamma_n(n) = \frac{\gamma_n}{n}\left(1-\lambda_n\right) \tag{1.84b}$$

With the two derivatives, equation (**??**), p. 36 can be rewritten into the following equivalent formula:

$$\vec{J}_n = \mu_n \left(qn\vec{E}_n + k_b T\nabla n - nk_b T\nabla\left(\ln\gamma_n\right)\right) \tag{1.85}$$

The last term is the modification to Einstein relationship, which can be combinated into potential term. As a result, the current equation (**??**), p. 11 keeps unchanged, but the effective driving force should be modified as:

$$\vec{E}_n = \frac{1}{q}\nabla E_c - \frac{k_b T}{q}\nabla\left(\ln(N_c) - \ln(T^{3/2})\right) - \frac{k_b T}{q}\nabla\left(\ln\gamma_n\right) \tag{1.86}$$

The same formula exists for holes:

$$\vec{E}_p = \frac{1}{q}\nabla E_v + \frac{k_b T}{q}\nabla\left(\ln(N_v) - \ln(T^{3/2})\right) + \frac{k_b T}{q}\nabla\left(\ln\gamma_p\right) \tag{1.87}$$

As a conclusion, when Fermi statistics is considered, the formula of DD method keeps unchanged, only an extra potential term should be considered. However, Fermi statistics also effect the implement of Ohmic boundary condition, please refer to [[TODO]]

CHAPTER 2

Genius Command Reference

Introduction

Format of Input Card

GENIUS code takes its input from a user specified card file. Each line of the card file is a particular statement, identified by the first keyword on the card. The remaining parts of the line are the parameters of that statement. The statement has the format as follow:

```
KEYWORD [parameters]
```

The words on a line should be separated by blanks or tabs. If more than one line of input is necessary for a particular statement, it may be continued on subsequent lines by placing a backslash sign \ as the last non-blank character on the current line.

GENIUS code parses the input file by GNU flex and bison. It first read the pattern file located at $GENIUS_DIR/lib/genius.key which provides the keyword names and parameter attributes before the parse work. After that, the grammar of user provide file is matched by this pattern. With this flexible mechanism, adding new statement to GENIUS is fairly easy.

Parameters may be one of five types: float, integer, bool, string or enumeration. The float point number supports C style double precision real number. The integer value also supports C style integer. The bool value can be True, On, False and Off. String value is made up of lower line, dot, blank, number and alpha characters. The string should not begin with number; quotation marks are only needed when string contains blank. The enumerate value is a string with its value in a limited range which is predefined in the pattern file. At last, the length of string (including enumerate string) is limited to 80 characters.

All the parameter specification has the same format as

```
parameter_name = [number|integer|bool|string|enumeration]
```

In the card descriptions, keywords, parameters, enumerate strings are not case sensitive. And their names does not need to be typed in full; only enough characters to ensure unique identification is necessary. However user input strings is case sensitive, because file name may be specified by the string. Comments must begin with '#' and can be either an separated line or locate at the end of current statement.

The sequence of input deck Most of the cards GENIUS used are sequence insensitive. The order of occurrence of cards is significant in only two cases. The mesh generation cards must have the right order, or it can't work properly. And GENIUS will execute the 'driven' cards sequentially. So the placement order of 'driven' cards will affect simulation result.

Statement Description Format

Syntax of Parameter Lists The following special characters are used in the formatted parameter list:

```
slanted shape <<num>>   - value of parameter in the indicated
   type
Square brackets [ ] - optional group
Vertical bar |      - alternate choice
Parentheses ( )     - group hierarchy
Braces { }          - group hierarchy with high level
```

Value Types Besides some string parameters which have fixed values, most of the parameters need a user defined value. A lower case letter in angle brackets represents a value of a given type. The following types of values are represented:

```
<<num>>  - double precision numerical value
<<int>>  - integer value
<<bool>> - boolean value
<<str>>  - string value
<<enum>> - enumeration value
<<a>>-prex - array of values
```

Global Specification

GLOBAL

The **GLOBAL** command defines some global simulation parameters such as unit scale and ambient temperature.

```
GLOBAL
  [ TExternal=<<num>> ] [ DopingScale=<<num>> ] [ Orientation=<<
     int>> ]
  { [ Z.Width=<<num>> ] | [ CylindricalMesh=<<bool>> ] }
  [ ResistiveMetal=<<bool>> ]
```

Parameters

Parameter	Type	Description	Default	Unit
Global Simulation Parameters				
TExternal	numerical	Ambient temperature external to the simulated device. Synonym: LatticeTemp	300	K
DopingScale	numerical	Doping concentration normalization factor. Physical quantities are normalized based on this factor to reduce numerical errors. Usually, a good choice of this value is the highest doping concentration in the device, although in some cases lower values are more desirable.	1e18	cm^{-3}
Z.Width	numerical	For a 2D mesh, the width of the device in the z-direction. Internally calculated current densities are multiplied by this width to produce the terminal currents. This parameter only affect 2D mesh.	1.0	μm
CylindricalMesh	bool	For a 2D mesh, Genius will consider it as cylindrically symmetric. Electrical current output is integrated along the angular direction thus in the unit of Amperes.	false	none
Orientation	integer	The crystalline orientation of the silicon substrate. Only 100, 110, and 111 are recognized.	100	none
ResistiveMetal	bool	Flag to enable resistive metal in device structure, which is introduced from version 1.7.0	false	none

Description

When several **GLOBAL** commands are present in the input file, and a parameter is set multiple times, only the last value takes effect.

Example

```
# device width in Z dimension. Unit:um
GLOBAL Z.Width = 2

# specify initial temperature of device. Unit:K
GLOBAL LatticeTemp = 3e2

# set carrier scale reference value
GLOBAL DopingScale = 1e16

# enable resistive metal
GLOBAL ResistiveMetal = true
```

Mesh Generation

GENIUS code has flexible mesh data structure which support various shapes of 2D and 3D elements. The only limit is the element should have circum-circle for 2D or circum-sphere for 3D to meet finite volume method used by GENIUS. The supported element shapes can be triangle and quadrangle for 2D, tetrahedron, prism and hexahedron for 3D. Please note that the triangle and tetrahedron can be any shape; however, quadrangle, prism and hexahedron should meet the in circuit or in sphere requirement. The GENIUS solver is designed as element shape independent, which means using mixed shapes of elements in mesh is possible.

Figure 2.1: 2D elements

Figure 2.2: 3D elements

GENIUS offers several build-in mesh generators to build both 2D and 3D mesh for semiconductor device. At present, they can only generate simulation mesh with unique mesh element. However, it is possible to import device model girded with hyper-mesh from third party mesh generator.

Coordinate System

All the mesh generators use Cartesian coordinate system, in which the top horizontal line has the minimal y coordinate, left vertical line has the minimal x coordinate and the front vertical line has the minimal z coordinate. In general, y coordinate is the direction vertical to silicon wafer.

Mesh Statements

The device mesh is specified by a series of mesh statements. They are listed as required input order:

- **MESH**
- **X.MESH**
- **Y.MESH**
- **Z.MESH**
- **ELIMATE** (optional)
- **SPREAD** or **SPREAD3D** (optional)
- **REGEION**
- **FACE** (optional)

Please note that the order in which statements appear is important. Changing the order changes the results. Generally, a mesh is specified by the following steps:

- Tell GENIUS which mesh generator will be used. (**MESH**)
- The mesh begins as a set of (nonuniformly) spaced x-, y- and z- lines comprising a simple hexahedron (**X.MESH**, **Y.MESH** and **Z.MESH**).
- Mesh lines may be terminated inside the device, and redundant nodes removed from the grid (**ELIMINATE**).
- The hexahedron can be distorted to track non-planar geometry or match the doping profile, although strongly non-planar structures are difficult to treat in this way (**SPREAD**, **SPREAD3D**).
- Material regions can be specified as a union of hexahedrons, completing the mesh specification (**REGION**).
- Boundary faces can be specified as surface rectangles when necessary (**FACE**).

MESH

This statement indicates the beginning of the mesh generation behavior.

```
MESH
  Type=(S_Tet4|S_Prism6|S_Hex8|S_Tri3|S_Quad4)
  [ Triangle=<<str>> ]  [ Tetgen=<<str>> ]
```

Parameters

Parameter	Type	Description	Default	Unit
Type	enum	Indicates which mesh generator is to be used. It can be one of **S_Tet4**, **S_Prism6**, **S_Hex8**, **S_Tri3** and **S_Quad4**.	none	none

Triangle	string	The parameters passed to triangular mesh generator for **S**$\mathbf{S_P}$**rism6** and $\mathbf{S_T}$**ri3** meshes. The triangular mesher recognizes the same set of parameters as the Triangle program. The detailed description of the available options can be found at Triangle's home page `http://www.cs.cmu.edu/~quake/triangle.html`.	pzADq30Q	none
TetGen	string	The parameters passed to the tetrahedral mesh generator Tetgen code, which is used for $\mathbf{S_T}$**et4** meshes. The detailed description of this string can be found at Tetgen's home page `http://tetgen.berlios.de`	pzAYq1.5	none
File.Prefix	string	File name prefix of the mesh files to be loaded in the simulator. Currently only works with $\mathbf{S_T}$**et4** meshes generated by Tetgen. For a file prefix foo, the files foo.node, foo.face and foo.ele will be read for node, face, and element data of the mesh, respectively.	none	none

Description

All simulations in Genius are performed in a discrete mesh. The available types of meshes are enumerated above, and illustrated in figure **??**, p. 41 and figure **??**, p. 41.

There are two alternative ways to generate the mesh structures. The user can use the **X.MESH**, **Y.MESH** and **Z.MESH** commands to specify the position of mesh lines, and have Genius to call the appropriate mesh generator programs behind the scene.

Alternatively, the user can define the device geometry with external tools, produce the mesh with an external mesh generator, and commands Genius to import from files containing the mesh structure. Currently Genius can import mesh files generated by Tetgen for tetrahedral $\mathbf{S_T}$**et4** meshes.

Example

```
MESH  Type=S_Tri3
MESH  Type=S_Tet4  Tetgen="pzAq1.5"
MESH  Type=S_Tet4  File.Prefix="foo"
```

X.MESH Y.MESH and Z.MESH

The **X.MESH**, **Y.MESH** and **Z.MESH** cards specify the location of lines of nodes in a hexahedral mesh (rectangular mesh for 2D) . The original mesh can be modified by following mesh cards like **ELIMINATE** and **SPREAD** or **SPREAD3D**.

```
X.MESH
   {  Width=<<num>>  |  (  X.Min=<<num>>  X.Max=<<num>>  )  }
   {  N.Spaces=<<int>>  [Ratio=<<num>>]  |  H1=<<num>>  [H2=<<num>>]  }
   [Min.Space=<<num>>]

Y.MESH
   {  Depth=<<num>>  |  (  Y.Min=<<num>>  Y.Max=<<num>>  )  }
   {  N.Spaces=<<int>>  [Ratio=<<num>>]  |  H1=<<num>>  [H2=<<num>>]  }
   [Min.Space=<<num>>]

Z.MESH
   {  Width=<<num>>  |  (  Z.Min=<<num>>  Z.Max=<<num>>  )  }
   {  N.Spaces=<<int>>  [Ratio=<<num>>]  |  H1=<<num>>  [H2=<<num>>]  }
   [Min.Space=<<num>>]
```

Parameters

Parameter	**Type**	**Description**	**Default**	**Unit**
Width	numerical	The distance of the grid section in x or z direction. This direction is parallel to silicon wafer.	none	μm
Depth	numerical	The distance of the grid section in y direction. This direction is vertical to silicon wafer.	none	μm
X.Min	numerical	The x location of the left edge of the grid section. synonym: **X.Left**. When more than one **X.Mesh** statements exist, the **X.Min** location of next mesh edge will follow most right of previous grid section automatically.	none	μm
X.Max	numerical	The x location of the right edge of the grid section. synonym: **X.Right**.	none	μm

Y.Min	numerical	The y location of the top edge of the grid section. synonym: **Y.Top**. When more than one **Y.Mesh** statements exist, the **Y.Min** location of next mesh edge will follow most bottom of previous grid section automatically.	none	μm
Y.Max	numerical	The y location of the bottom edge of the grid section. synonym: **Y.Bottom**.	none	μm
Z.Min	numerical	The z location of the front edge of the grid section. synonym: **Z.Front**. When more than one **Z.Mesh** statements exist, the **Z.Min** location of next mesh edge will follow most back of previous grid section automatically.	none	μm
Z.Max	numerical	The z location of the bottom edge of the grid section. synonym: **Z.Back**.	none	μm
N.Spaces	integer	The number of grid spaces in the grid section.	1	none
Ratio	numerical	The ratio between the sizes of adjacent grid spaces in the grid section. **Ratio** should usually lie between 0.667 and 1.5.	1.0	none
H1	numerical	The size of the grid space at the begin edge of the grid section.	none	μm
H2	numerical	The size of the grid space at the end edge of the grid section.	none	μm
Min.Space	numerical	The size of the minimum allowed grid spacing in this grid section.	1e-4	μm

Description

[give pictures here]

Example

```
X.MESH    X.MIN=0.0   X.MAX=0.50   N.SPACES=8
Y.MESH    DEPTH=0.1   N.SPACES=8   RATIO=0.8
Y.MESH    DEPTH=0.1   N.SPACES=20
Y.MESH    DEPTH=0.6   H1=0.005   H2=0.050
Z.MESH    WIDTH=1.0   N.SPACES=5
```

ELIMINATE

The **ELIMINATE** statement eliminates mesh points along planes in a hexahedral grid over a specified volume. This statement is useful for eliminating nodes in regions of the device structure where the grid is more dense than necessary. Points along every second line in the chosen direction within the chosen range are removed, except the first and last line. Successive eliminations of the same range remove points along every fourth line, eighth line, and so on.

```
ELIMINATE
  { Direction = (XNorm | YNorm | ZNorm) }
  [ {X.Min=<<num>> | IX.Min=<<int>>} ] [ {X.Max=<<num>> | IX.Max
     =<<int>>} ]
  [ {Y.Min=<<num>> | IY.Min=<<int>>} ] [ {Y.Max=<<num>> | IY.Max
     =<<int>>} ]
  [ {Z.Min=<<num>> | IZ.Min=<<int>>} ] [ {Z.Max=<<num>> | IZ.Max
     =<<int>>} ]
```

Parameters

Parameter	**Type**	**Description**	**Default**	**Unit**
Direction	enum	Specifies that lines of nodes in direction of X, Y or Z are eliminated.	none	none
X.Min	numerical	The minimum x location of the hexahedron volume in which nodes are eliminated. The default value is the minimum x-coordinates in the device. synonym: **X.Left**.	XMin	μm
X.Max	numerical	The maximum x location of the hexahedron volume in which nodes are eliminated. The default value is the maximum x-coordinates in the device. synonym: **X.Right**.	XMax	μm
IX.Min	integer	The minimum x node index of the hexahedron volume in which nodes are eliminated. synonym: **IX.Left**.	0	none
IX.Max	integer	The maximum x node index of the hexahedron volume in which nodes are eliminated. synonym: **IX.Right**.	IXMax-1	none

Y.Min	numerical	The minimum y location of the hexahedron volume in which nodes are eliminated. The default value is the minimum y-coordinates in the device. synonym: **Y.Top**.	YMin	μm
Y.Max	numerical	The maximum y location of the hexahedron volume in which nodes are eliminated. The default value is the maximum y-coordinates in the device. synonym: **Y.Bottom**.	YMax	μm
IY.Min	integer	The minimum y node index of the hexahedron volume in which nodes are eliminated. synonym: **IY.Top**.	0	none
IY.Max	integer	The maximum y node index of the hexahedron volume in which nodes are eliminated. synonym: **IY.Bottom**.	IYMax-1	none
Z.Min	numerical	The minimum Z location of the hexahedron volume in which nodes are eliminated. The default value is the minimum Z-coordinates in the device. synonym: **Z.Front**.	ZMin	μm
Z.Max	numerical	The maximum z location of the hexahedron volume in which nodes are eliminated. The default value is the maximum z-coordinates in the device. synonym: **Z.Back**.	ZMax	μm
IZ.Min	integer	The minimum z node index of the hexahedron volume in which nodes are eliminated. synonym: **IZ.Front**.	0	none
IZ.Max	integer	The maximum x node index of the hexahedron volume in which nodes are eliminated. synonym: **IZ.Back**.	IZMax-1	none

Description

Not all the mesh generator support **ELIMINATE**. The $\mathbf{S_QUAD4}$ and $\mathbf{S_HEX8}$ mesh generator can not do mesh line eliminate. User should take care of the elimination process. The excessively elimination may cause bad element quality.

Example

```
ELIMINATE    Direction=YNORM    Y.TOP=-1.0
ELIMINATE    Direction=XNORM    IX.MAX=8
```

SPREAD

The **SPREAD** statements provide a way to adjust the y position of nodes along grid lines parallel to the x-plan (the disturb is uniform in Z dimension) in a hexahedral mesh to follow surface and junction contours. The statement is only supported by **$S_T$ri3**, **$S_T$et4** and **$S_P$rism6** mesh generator.

```
SPREAD
  Location = (Left|Right) Width=<<num>> Upper=<<int>> Lower=<<
     int>>
  [ Enchroach=<<num>>]
  { Y.Lower=<<num>> | (Thickness=<<num>> [Vol.Rat=<<num>>]) }
  [ Grading=<<num>> ]
```

Parameters

Parameter	Type	Description	Default	Unit
Location	enum	Specifies which side of the grid is distorted.	none	none
Width	numerical	The width of the distorted region measured from the left or right edge of the structure.	0	μm
Upper	integer	The index of the upper y-grid line of the distorted region.	0	none
Lower	integer	The index of the lower y-grid line of the distorted region.	0	none
Encroach	integer	The factor which defines the abruptness of the transition between distorted and undistorted grid. The transition region becomes more abrupt with smaller **Encroach** factors. The minimum allowed value is 0.1.	1.0	none
Y.Lower	numerical	The vertical location in the distorted region where the line specified by **Lower** is moved. The grid line specified by **Upper** does not move if this parameter is specified.	none	μm

Thickness	numerical	The thickness of the distorted region. Specifying **Thickness** usually causes the positions of both the **Upper** and **Lower** grid lines to move.	none	μm
Vol.Rat	numerical	The ratio of the displacement of the lower grid line to the net change in thickness. If **Vol.Rat** is 0, the location of the lower grid line does not move. If **Vol.Rat** is 1, the upper grid line does not move.	0.44	none
Grading	numerical	The vertical grid spacing ratio in the distorted region between the y-grid lines specified with **Upper** and **Lower**. The spacing grows or shrinks by **Grading** in each interval between lines. **Grading** should usually lie between 0.667 and 1.5.	1.0	none

Example

```
SPREAD    Location=Left   Width=0.625  Upper=0  Lower=2  \
          Thickness=0.1

SPREAD    Location=Right  Width=0.625  Upper=0  Lower=2  \
          Thickness=0.1
```

SPREAD3D

The **SPREAD3D** statements also provide a way to adjust the y position of nodes along grid lines parallel to the xz-plan in a hexahedral mesh to follow surface and junction contours. The statement is only supported by S_Tet4 mesh generator.

```
SPREAD3D
  [ {X.Min=<<num>>  |  IX.Min=<<int>>}  ]  [  {X.Max=<<num>>  |  IX.Max
     =<<int>>}  ]
  [ {Z.Min=<<num>>  |  IZ.Min=<<int>>}  ]  [  {Z.Max=<<num>>  |  IZ.Max
     =<<int>>}  ]
  Upper=<<int>>  Lower=<<int>>  [  Enchroach=<<num>>]
  {  Y.Lower=<<num>>   |  (Thickness=<<num>>  [Vol.Rat=<<num>>])  }
  [Grading=<<num>>]
```

Parameters

Parameter	**Type**	**Description**	**Default**	**Unit**
X.Min	numerical	The minimum x location of the area in which nodes are disturbed. synonym: **X.Left**.	XMIN	μm
X.Max	numerical	The maximum x location of the area in which nodes are disturbed. synonym: **X.Right**.	XMAX	μm
IX.Min	integer	The minimum x node index of the area in which nodes are disturbed. synonym: **IX.Left**.	0	none
IX.Max	integer	The maximum x node index of the area in which nodes are disturbed. synonym: **IX.Right**.	IXMAX-1	none
Z.Min	numerical	The minimum z location of the area in which nodes are disturbed. synonym: **Z.Front**.	ZMIN	μm
Z.Max	numerical	The maximum z location of the area in which nodes are disturbed. synonym: **Z.Back**.	ZMAX	μm
IZ.Min	integer	The minimum z node index of the area in which nodes are disturbed. synonym: **IZ.Front**.	0	none
IZ.Max	integer	The maximum z node index of the area in which nodes are disturbed. synonym: **IZ.Back**.	IZMAX-1	none
Upper	integer	The index of the upper y-grid line of the distorted region.	0	none
Lower	integer	The index of the lower y-grid line of the distorted region.	0	none
Encroach	integer	The factor which defines the abruptness of the transition between distorted and undistorted grid. The transition region becomes more abrupt with smaller **Encroach** factors. The minimum allowed value is 0.1.	1.0	none
Y.Lower	numerical	The vertical location in the distorted region where the line specified by **Lower** is moved. The grid line specified by **Upper** does not move if this parameter is specified.	none	μm

Thickness	numerical	The thickness of the distorted region. Specifying **Thickness** usually causes the positions of both the **Upper** and **Lower** grid lines to move.	none	μm
Vol.Rat	numerical	The ratio of the displacement of the lower grid line to the net change in thickness. If **Vol.Rat** is 0, the location of the lower grid line does not move. If **Vol.Rat** is 1, the upper grid line does not move.	0.44	none
Grading	numerical	The vertical grid spacing ratio in the distorted region between the y-grid lines specified with **Upper** and **Lower**. The spacing grows or shrinks by **Grading** in each interval between lines. **Grading** should usually lie between 0.667 and 1.5.	1.0	none

Example

```
SPREAD3D      X.MIN=1.0  X.MAX=2.0  Z.MIN=0.0  Z.MAX=1.0    \
              Upper=0  Lower=2  Thickness=0.1
```

REGION

The **REGION** statement defines the location of materials in the mesh. Currently, GENIUS supports following materials: Vacuum; semiconductor material including Si, Ge, GaAs, 3C-SiC, Si1-xGex, AlxGa1-xAs, InxGa1-xAs, InP, InAs, InN, InSb, Hg(1-x)Cd(x)Te; insulator material including Air, SiO2, Si3N4 and electrode region including Elec, Al, Cu, Au, Ag and PolySi.

```
REGION
  Shape=(Rectangle|Ellipse)  Label=<<str>>  Material=<<str>>
  [ {X.Min=<<num>>  |  IX.Min=<<int>>}  ] [ {X.Max=<<num>>  |  IX.Max
     =<<int>>}  ]
  [ {Y.Min=<<num>>  |  IY.Min=<<int>>}  ] [ {Y.Max=<<num>>  |  IY.Max
     =<<int>>}  ]
  [ {Z.Min=<<num>>  |  IZ.Min=<<int>>}  ] [ {Z.Max=<<num>>  |  IZ.Max
     =<<int>>}  ]
```

Parameters

Parameter	Type	Description	Default	Unit

Shape	enum	Specifies the shape of the region. Can be **Rectangle** or **Ellipse**.	**Rectangle**	none
Label	string	Specifies the identifier of this region, limited to 12 chars.	none	none
Material	string	Specifies the material of the region.	none	none
X.Min	numerical	The minimum x location of the region. synonym: **X.Left**.	XMIN	μm
X.Max	numerical	The maximum x location of the region. synonym: **X.Right**.	XMAX	μm
IX.Min	integer	The minimum x node index of the region. synonym: **IX.Left**.	0	none
IX.Max	integer	The maximum x node index of the region. synonym: **IX.Right**.	IXMAX-1	none
Y.Min	numerical	The minimum y location of the region. synonym: **Y.Top**.	YMIN	μm
Y.Max	numerical	The maximum y location of the region. synonym: **Y.Bottom**.	YMAX	μm
IY.Min	integer	The minimum y node index of the region. synonym: **IY.Top**.	0	none
IY.Max	integer	The maximum y node index of the region. synonym: **IY.Bottom**.	IYMAX-1	none
Z.Min	numerical	The minimum z location of the region. synonym: **Z.Front**.	ZMIN	μm
Z.Max	numerical	The maximum z location of the region. synonym: **Z.Back**.	ZMAX	μm
IZ.Min	integer	The minimum z node index of the region. synonym: **IZ.Front**.	0	none
IZ.Max	integer	The maximum z node index of the region. synonym: **IZ.Back**.	IZMAX-1	none

Description

Several regions can be defined one by one. But users should be careful that regions can't get cross each other. The situations showed in figure **??**, p. 52(A) and (B) are allowed, but (C) will break the mesh generator of genius.

Figure 2.3: Multi-Region definition.

When the background grid line is disturbed by **SPREAD** statement, it is something difficult for specifying region bounding with grid line location. One can use grid line index instead of grid line location here.

Example

```
REGION   Label=Si1     Material=Si    Y.TOP= 0.0  Y.BOTTOM=−0.100
REGION   Label=SiGe1   Material=SiGe  Y.TOP=−0.1  Y.BOTTOM=−0.125
```

FACE

Face is a group of facet elements (triangle and quadrangle for 3D, and edge for 2D) which have the same attribute. This statement specifies the label of a special face. User can assign the face with a special boundary condition type by **BOUNDARY** statement.

```
FACE
  Label=<<str>>
  { Location = (Top|Bottom|Left|Right|Front|Back) |
    ( Direction=(XNorm|YNorm|ZNorm) ( X=<<num>>|Y=<<num>>|Z=<<
       num>> ) ) }
  [ {X.Min=<<num>> | IX.Min=<<int>>} ] [ {X.Max=<<num>> | IX.Max
     =<<int>>} ]
  [ {Y.Min=<<num>> | IY.Min=<<int>>} ] [ {Y.Max=<<num>> | IY.Max
     =<<int>>} ]
  [ {Z.Min=<<num>> | IZ.Min=<<int>>} ] [ {Z.Max=<<num>> | IZ.Max
     =<<int>>} ]
```

Parameters

Parameter	**Type**	**Description**	**Default**	**Unit**
Label	string	Specifies the identifier of this face, limited to 80 chars.	none	none
Location	enum	Specifies which side the face lies along.	none	none
Direction	enum	Specifies the dimensional orientation of the face.	none	none
X	numerical	Specifies the X coordinate of the face parallel to YZ plane.	0.0	μm
Y	numerical	Specifies the Y coordinate of the face parallel to XZ plane.	0.0	μm
Z	numerical	Specifies the Z coordinate of the face parallel to XY plane.	0.0	μm
X.Min	numerical	The minimum x location of the face. synonym: **X.Left**.	XMIN	μm
X.Max	numerical	The maximum x location of the face. synonym: **X.Right**.	XMAX	μm

IX.Min	integer	The minimum x node index of the face. synonym: **IX.Left**.	0	none
IX.Max	integer	The maximum x node index of the face. synonym: **IX.Right**.	IXMAX-1	none
Y.Min	numerical	The minimum y location of the face. synonym: **Y.Top**.	YMIN	μm
Y.Max	numerical	The maximum y location of the face. synonym: **Y.Bottom**.	YMAX	μm
IY.Min	integer	The minimum y node index of the face. synonym: **IY.Top**.	0	none
IY.Max	integer	The maximum y node index of the face. synonym: **IY.Bottom**.	IYMAX-1	none
Z.Min	numerical	The minimum z location of the face. synonym: **Z.Front**.	ZMIN	μm
Z.Max	numerical	The maximum z location of the face. synonym: **Z.Back**.	ZMAX	μm
IZ.Min	integer	The minimum z node index of the face. synonym: **IZ.Front**.	0	none
IZ.Max	integer	The maximum z node index of the face. synonym: **IZ.Back**.	IZMAX-1	none

Description

The **FACE** statement can specify facet elements not only on the boundary of the mesh, but also inside the mesh, i.e. at the interface of two regions. Here gives the rules about how GENIUS dealing with of faces. The faces specified by **FACE** statement always has the highest priority. They will have the label as **FACE** statement declared. For the remaining faces, the interface faces between two regions will be assigned by name1_to_name2 in which the name1 and name2 are the labels of the two regions in alphabetic order. The remain faces of a region will be assigned by name_Neumann and the name is the label of the region. There are some limitations about **Sprism6** mesh generator. Face can not be defined along Z direction, i.e. on front or back plane.

Example

```
FACE   Label=Anode     Direction=Horizontal  X.MIN=0.0  X.MAX=1.0  \
       Z.MIN=0.0  Z.MAX=3.0  Y=0.0

FACE   Label=Cathode  Direction=Horizontal  X.MIN=0.0  X.MAX=3.0  \
       Z.MIN=0.0  Z.MAX=3.0  Y=3.0

FACE   Label=Anode      Location=TOP     X.MIN=0.0  X.MAX=1.0

FACE   Label=Cathode    Location=BOTTOM
```

REFINE

The **REFINE** statement allows refinement of a coarse mesh. GENIUS supports three types of mesh refinement: uniform, conform and hierarchical mesh refinement. At present, the solution is destroyed after the mesh refinement. Thus user should ensure that there are **DOPING** and probably **MOLE** statements exist in the same input file so that device doping profile and mole distribution can be rebuild on the new mesh.

```
REFINE.Uniform   [Step=int]

REFINE.Conform
  Variable = ( Doping | Potential | Electron | Hole |
               Temperature | E.Temp | H.Temp | QFN | QFP |
               E.Field | Net.Carrier | Net.Charge |
               Optical.Gen | Particle.Gen )
  Evaluation = ( Gradient | Quantity )
  Measure=( Linear| SignedLog ) [ Region=<<str>> ]
  { error.fraction=<<num>> | cell.fraction=<<num>> |
    error.threshold=<<num>> }

REFINE.Hierarchical
  Variable = ( Doping | Potential | Electron | Hole |
               Temperature | E.Temp | H.Temp | QFN | QFP |
               E.Field | Net.Carrier | Net.Charge )
  Evaluation = ( Gradient | Quantity )
  Measure=( Linear| SignedLog ) [ Region=<<str>> ]
  { error.refine.fraction=<<num>>  | cell.refine.fraction=<<num
     >>  |
    error.refine.threshold=<<num>> }
  [ error.coarsen.fraction=<<num>> | cell.coarsen.fraction=<<num
     >> |
    error.coarsen.threshold=<<num>> ]
```

Parameters

Parameter	Type	Description	Default	Unit
Uniform Mesh Refine				
Step	integer	how many steps should the uniform refinement repeats.	none	none
General Mesh Refinement Parameters				
Variable	enum	the mesh refinement criteria variable. Can be one of: Doping, Potential, Electron, Hole, Temperature, E.Temp, H.Temp, QFN, QFP, E.Field, Net.Carrier and Net.Charge.	none	none

Evaluation	enum	Specifies whether the variable quantity or its gradient should be used in refinement critiria.	**Gradient**	none
Measure	enum	Specifies that refinement is based on the original value or logarithm of the specified quantity.	**Linear**	none
Region	string	The name of the region over which refinement takes place. Elements in other regions may be refined for mesh conform or well-shaped elements. User can specify several regions by multi Region parameters.	none	none
Conformal Mesh Refinement				
Error.Fraction	numerical	When the cell's error exceeds $\mathbf{error.fraction} * \mathbf{max_error}$, it will be refined..	0.3	none
Error.Threshold	numerical	When the cell's error exceeds **error.threshold**, it will be refined.	0.1	none
Cell.Fraction	numerical	The $\mathbf{cell.fraction} * (\mathbf{totalcellnumber})$ of cells with most error will be refined.	0.3	none
X.Min	numerical	The minimum x coordinate of the region that will be considered for refinement. synonym: **X.Left**.	0	μm
X.Max	numerical	The maximum x coordinate of the region that will be considered for refinement. synonym: **X.Right**.	0	μm
Y.Min	numerical	The minimum y coordinate of the region that will be considered for refinement. synonym: **Y.Top**.	0	μm
Y.Max	numerical	The maximum y coordinate of the region that will be considered for refinement. synonym: **Y.Bottom**.	0	μm
Z.Min	numerical	The minimum z coordinate of the region that will be considered for refinement. synonym: **Z.Front**.	0	μm

Z.Max	numerical	The maximum z coordinate of the region that will be considered for refinement. synonym: **Z.Back**.	0	μm
Hierarchical Mesh Refinement/Coarsening				
Error.Refine.Fr	numerical	When the cell's error exceeds **error.refine.fraction** $*$ $\mathbf{max_error}$, it will be refined. synonym: Error.Refine.Fraction	0.3	none
Error.Coarsen.Fr	numerical	When the cell's error is less than **error.coarsen.fraction** $*$ $\mathbf{max_error}$, it will be coarsened. synonym: **Error.Coarsen.Fraction**	0.0	none
Error.Refine.Th	numerical	When the cell's error exceeds **error.refine.threshold**, it will be refined. synonym: **Error.Refine.Threshold**	0.1	none
Error.Coarsen.Th	numerical	When the cell's error is less than **error.coarsen.threshold**, it will be coarsened. synonym: **Error.Coarsen.Threshold**	0.1	none
Cell.Refine.Fr	numerical	The **cell.refine.fraction** $*$ (**totalcellnumber**) of cells with most error will be refined. synonym: **Cell.Refine.Fraction**	0.3	none
Cell.Coarsen.Fr	numerical	The **cell.coarsen.fraction** $*$ (**totalcellnumber**) of cells with least error will be coarsened. synonym: **Cell.Coarsen.Fraction**	0.3	none

Description

REFINE.Uniform is only intended for debugging. Only **TRI3** and **TET4** mesh can do **REFINE.Conform**. And the conform refine process requires the mesh should be generated by corresponding mesh generator which is defined in the same input file. However, this is the most stable mesh refinement procedure. The **REFINE.Hierarchical** is the new mesh refine and coarsen mechanism introduced into GENIUS. It supports all the elements and regardless mesh generators. This method will generate hanging node on the side of elements with different refinement level. User should ensure that the hanging node be far away from pn junction of the device. Or convergence problem may occur. Generally, refine by potential has the best effect.

Example

```
REFINE.U  Step=1

REFINE.C  Variable=potential  cell.fraction=0.3

REFINE.H  Variable=potential  cell.refine.fraction=0.3  \
          cell.coarsen.fraction=0.1
```

External Sources

For simulation the transient response of device, GENIUS supports several types of voltage and current source. The original models of these sources come from SPICE, a famous circuit simulation program. Several sources may be defined in one disk file. And the placement of these definitions are not critical. The sources can be assigned to electrode by **ATTACH** statement when needed.

VSOURCE

The **VSOURCE** command defines a voltage source.

```
VSOURCE
  ID=str    [ TDelay = num ]
  [ Type = (VDC|VSin|VExp|VPulse|VShell) ]

DC Source
  { VConst=num } |

Sinusoidal Source
  { VAmp=num VFreq=num } |

Exponential Source
  { VHi=num VLo=num TRC=num TFD=num TFC=num  } |

Pulse Source
  { VHi=num VLo=num Tr=num Tf=num Pw=num Pr=num } |

User-defined Source
  { DLL=str Function=str }
```

Parameters

Parameter	Type	Description	Default	Unit
General Parameters				

Type	enum	Selects the type of voltage source. The following types are available: DC source, sinusoidal source, exponential source, pulse source and user-defined source.	VDC	none
ID	string	Declares the name of the source.	none	none
TDelay	numerical	Delay time before the source voltage is applied.	none	none
DC Source Parameters				
VConst	numerical	Constant voltage.	0.0	V
Sinusoidal Source Parameters				
VAmp	numerical	Amplitude of the sinusoidal source.	0.0	V
VFreq	numerical	Frequency of the sinusoidal source	none	Hz
Exponential Source Parameters				
VHi	numerical	Upper bound of exponential voltage waveform.	0.0	V
VLo	numerical	Lower bound of exponential voltage waveform.	0.0	V
TRC	numerical	Rising edge time constant.	none	s
TFD	numerical	Delay time before the falling phase starts. The length of the rising phase is thus TFD-TDelay.	none	s
TFC	numerical	Falling edge time constant.	none	s
Pulse Source Parameters				
VHi	numerical	High-voltage level of the pulses.	0.0	V
VLo	numerical	Lower-voltage level of the pulses.	0.0	V
Tr	numerical	Rising edge time constant.	1e-12	s
Tf	numerical	Falling edge time constant.	1e-12	s
Pw	numerical	Pulse width.	0.0	s
Pr	numerical	Repeating period of the pulse train.	0.0	s
User-defined Source Parameters				
DLL	string	Name of the dynamically loadable library file that contains the user-defined source.	none	none
Function	string	Function name of the source in the DLL file.	none	none

Description

GENIUS supports user defined voltage and current source by loading shared object (.so) file. The file which contains a user defined voltage source should have the function as follow. GENIUS will pass the argument time in the unit of second to the function vsrc_name and get voltage value in the unit of volt. The current source function is almost the same except the unit of current is A.

```
double vsrc_name(double time)
{
   /* calculate the voltage amplitude */
   return vsrc_amplitude;
}
double isrc_name(double time)
{
    /* calculate the current amplitude */
    return isrc_amplitude;
}
```

The c code should be linked with −**shared** and −**fPIC** option as:

```
gcc -shared -fPIC -o foo.so foo.c -lm
```

The *foo.so* file should be put in the same directory as input file.

Example

```
vsource Type=VDC     ID=GND   Tdelay=0 Vconst=0

vsource Type=VDC     ID=VCC   Tdelay=0 Vconst=5

vsource Type=VSIN    ID=Vs    Tdelay=1e-6 Vamp=0.1 Freq=1e6

vsource Type=VEXP    ID=V1    Tdelay=0 TRC=1e-6 TFD=1e-6 \
        TFC=1e-6 Vlo=0 Vhi=1

vsource Type=VPULSE  ID=V2    Tdelay=0 Tr=1e-9 Tf=1e-9 \
        Pw=5e-6 Pr=1e-5 Vlo=0 Vhi=1

vsource Type=VSHELL  ID=VGauss  DLL=foo.so Func=vsrc_gauss
```

ISOURCE

The **ISOURCE** command defines a voltage source.

```
ISOURCE
  ID=str  [ Type = (IDC|ISin|IExp|IPulse|IShell) ]
  [ TDelay = num ]

DC Source
  { IConst=num } |
Sinusoidal Source
  { IAmp=num IFreq=num } |

Exponential Source
  { IHi=num ILo=num TRC=num TFD=num TFC=num  } |

Pulse Source
  { IHi=num ILo=num Tr=num Tf=num Pw=num Pr=num } |

User-defined Source
  { DLL=str Function=str }
```

Parameters

Parameter	Type	Description	Default	Unit
General Parameters				
Type	enum	Selects the type of current source. The following types are available: DC source, sinusoidal source, exponential source, pulse source and user-defined source.	VDC	none
ID	string	Declares the name of the source.	none	none
TDelay	numerical	Delay time before the source is applied.	none	none
DC Source Parameters				
IConst	numerical	Constant current.	0.0	V
Sinusoidal Source Parameters				
IAmp	numerical	Amplitude of the sinusoidal source.	0.0	V
IFreq	numerical	Frequency of the sinusoidal source	none	Hz
Exponential Source Parameters				
IHi	numerical	Upper bound of exponential current waveform.	0.0	V
ILo	numerical	Lower bound of exponential current waveform.	0.0	V
TRC	numerical	Rising edge time constant.	none	s
TFD	numerical	Delay time before the falling phase starts. The length of the rising phase is thus TFD-TDelay.	none	s
TFC	numerical	Falling edge time constant.	none	s
Pulse Source Parameters				
IHi	numerical	High-current level of the pulses.	0.0	V

ILo	numerical	Lower-current level of the pulses.	0.0	V
Tr	numerical	Rising edge time constant.	1e-12	s
Tf	numerical	Falling edge time constant.	1e-12	s
Pw	numerical	Pulse width.	0.0	s
Pr	numerical	Repeating period of the pulse train.	0.0	s
User-defined Source Parameters				
DLL	string	Name of the dynamically loadable library file that contains the user-defined source.	none	none
Function	string	Function name of the source in the DLL file.	none	none

Example

```
ISource  Type=IDC     ID=I1  Tdelay=0  Iconst=5

Isource  Type=ISIN    ID=I2  Tdelay=0  Iamp=0.1  Freq=1e6

Isource  Type=IEXP    ID=I3  Tdelay=0  TRC=1E-6  TFD=3E-6  \
         TFC=1E-6  Ilo=0  Ihi=1

Isource  Type=IPULSE  ID=I4  Tdelay=0  Tr=1E-9  Tf=1E-9  \
         Pw=5E-6  Pr=1E-5  Ilo=0  Ihi=1
```

PARTICLE

```
PARTICLE

From scatter data file
  Profile = ( FromFile2D | FromFile3D ) Profile.File=<<str>>
  [Transform.xx=<<num>>] [Transform.xy=<<num>>] [Transform.xz=<<
     num>>]
  [Transform.yx=<<num>>] [Transform.yy=<<num>>] [Transform.yz=<<
     num>>]
  [Transform.zx=<<num>>] [Transform.zy=<<num>>] [Transform.zz=<<
     num>>]
  [Translate.x=<<num>>]  [Translate.y=<<num>>]  [Translate.z=<<
     num>>]

From analytic expression
  Profile = Expression LET=<<num>>

From particle track
  Profile = Track  Profile.File=<<str>>

  [Energy.Eff=<<num>>]   [t0=<<num>> ] [tmax=<<num>>] [ t.Char
     =<<num>> ]
```

Parameters

Parameter	**Type**	**Description**	**Default**	**Unit**
Profile	enum	Type of particle profile input.	none	none
Transform	numerical	The transformation matrix applied to the coordinates imported from the input file. The final coordinates are obtained from Transform*location + Translate.	I	none
Translate	enum	The translation vector applied to the input coordinates.	0	none
Energy.Eff	numerical	The energy loss for each generated electron-hole pair.	3.6	eV
t0	numerical	The starting time when high-energy particles penetrates the device.	0.0	s
tmax	numerical	The time instant when electron-hole pair generation rate reaches its maximum.	0.0	s
t.Char	numerical	The variance (σ) of the Gaussian decay of carrier generation rate.	1e-12	s

Description

There are several ways to specify the energy deposit of a particle: by scatter data file, by analytic express and by particle track.

The input data file must be in plain text, with numerical values separated by white spaces. Each line contains the coordinates and the energy deposition at that position. For 2D profile, each line has 3 values, while for 3D profile, each line has 4 values. The coordinates are in the unit of μm, while the energy deposition is in the unit of $eV \mu m^{-3}$.

Examples

```
PARTICLE Profile=FromFile3D Profile.File="particle.dat" \
         tmax=2e-12 t.char=2e-12
```

LIGHT

```
LIGHT
  { SpectrumFile=<<str>> | ( Lambda=<<num>> Intensity=<<num>> [
      Eta=<<num>>] ) }
```

```
Profile = ( FromFile2D | FromFile3D )
[ Profile.File=<<str>> ] [ SkipLine=<<num>> ]
[ LUnit = (m|cm|um|nm) ] [ FUnit = (m|cm|um|nm) ]
[Transform.xx=<<num>>] [Transform.xy=<<num>>] [Transform.xz=<<
    num>>]
[Transform.yx=<<num>>] [Transform.yy=<<num>>] [Transform.yz=<<
    num>>]
[Transform.zx=<<num>>] [Transform.zy=<<num>>] [Transform.zz=<<
    num>>]
[Translate.x=<<num>>]  [Translate.y=<<num>>]  [Translate.z=<<
    num>>]
```

Parameters

Parameter	**Type**	**Description**	**Default**	**Unit**
Profile	enum	Type of field profile input.	**none**	none
Lambda	numerical	Wavelength of the monochromatic light.	**0.532**	μm
Intensity	numerical	Intensity of the monochromatic light.	**0.0**	W/cm^2
Eta	numerical	Quantum efficiency of carrier generation for the monochromatic light. If not explicitly specified, Genius will automatically calculate the quantum efficiency using the bandgap of the material.	**auto**	none
SpectrumFile	string	Filename of the spectrum file. This file should describe the wavelength and intensity at each frequency in the spectrum.	**none**	none
Profile.File	enum	Filename of the field solution data file. When a spectrum file is provided, the provided filename extension is appended to the field solution filename.	**none**	none
SkipLine	integer	Skip the first **SkipLine** number of lines in the field data file.	0	none
LUnit	enum	Length unit used for node coordinates in the field data file.	**um**	none
FUnit	enum	Length unit used in E-field value. The voltage unit is always Volt.	**um**	none

Transform	numerical	The transformation matrix applied to the coordinates imported from the input file. The final coordinates are obtained from **Transform***location + **Translate**.	I	none
Translate	enum	The translation vector applied to the input coordinates.	0	none

Description

Spectrum file

With the spectrum file we can load the field data at multiple wavelengths in the same **LIGHT** command. Each line in the spectrum file should contain the following columns

- A filename extension for the field data file. Suppose **Profile.File** is *diode1*, and the extension is *3290*, Genius will try to load the field data from the file *diode1.3290*.
- The wavelength in the unit of micro-meter.
- Intensity density (per unit wavelength) at this wavelength. The unit is W/cm^2/ μm. Note that this differs from the **intensity** parameter for monochromatic light.
- (Optional) quantum efficiency for carrier generation. If not explicitly specified, Genius will automatically calculate the quantum efficiency using the bandgap of the material.

A sample spectrum file is listed below.

```
# ext     lambda     intensity
3290      0.912      0.2
3145      0.954      0.5
3000      1.00       0.4
2855      1.05       0.3
2710      1.11       0.2
2565      1.17       0.2
2420      1.24       0.1
```

Optical Carrier Generation

HFSS assumes by default that the incident wave has E-field strength of $E_0 = 1V/m$, which corresponds to the incident intensity of

$$P_0 = \frac{1}{2}\epsilon_0 c_0 {E_0}^2. \tag{2.1}$$

We need to scale the E-field in the solution according to the user-specified incident intensity.

If the user does not supply the quantum efficiency value, Genius will calculate with the following assumption

$$\eta = floor(\frac{E_g}{h\nu}). \tag{2.2}$$

where the *floor* function round to largest integral value not greater than the argument. The carrier generation is finally calculated with

$$G = \eta \frac{\pi \epsilon_0 \left(-\epsilon_r''\right)}{h} E^2. \tag{2.3}$$

HFSS places a descriptive heading in each spectrum file, we shall skip that line.

Field Data File

The input data file must be in plain text, with numerical values separated by white spaces. Each line contains the coordinates and the E-field strength of the EM wave at that position. For 2D profile, each line has 3 values, while for 3D profile, each line has 4 values. The unit of the coordinates is specified by **LUnit**, and the unit of the E-field is specified by **FUnit**.

Examples

```
LIGHT       SpectrumFile=diode1  Profile.File=diode1  \
            Profile=fromfile3d  LUnit=m  FUnit=m  SkipLine=1
```

ENVELOP

The **ENVELOP** command defines an envelop in time, which is used to modulate light source. The light source can be introduced by **LIGHT** command and **RAYTRACE** command. User can specify the envelop of light source at the **SOLVE** command.

```
ENVELOP
  ID = str
  [ Type = (Uniform|Gaussian|Pulse|Expression|Shell) ]
  [ TDelay = num ]

Uniform envelop
  { Amplitude=num } |

Gaussian envelop
  { Amplitude=num T0=num Tao=num} |

Pulse envelop
  { Amplitude.high=num Amplitude.low=num
    Tr=num Tf=num Pw=num Pr=num } |

User-defined envelop by expression
  { Expression=str }
```

```
User−defined envelop by dynamic loadable library
  { DLL=str Function=str }
```

Parameters

Parameter	Type	Description	Default	Unit
General Parameters				
Type	enum	Selects the type of envelop. The following types are available: Uniform, Gaussian, Pulse and user-defined envelop.	Uniform	none
ID	string	Declares the name of the envelop.	none	none
TDelay	numerical	Delay time before the envelop is applied.	none	none
Uniform Envelop Parameters				
Amplitude	numerical	Constant amplitude.	1.0	none
Gaussian Envelop Parameters				
Amplitude	numerical	Amplitude of Gaussian waveform.	1.0	none
T0	numerical	The time when Gaussian waveform reaches its max.	0.0	s
Tao	numerical	The characteristic time of Gaussian waveform.	1e-12	s
Pulse Envelop Parameters				
Amplitude.high	numerical	High level of the pulses.	1.0	V
Amplitude.low	numerical	Lower level of the pulses.	0.0	V
Tr	numerical	Rising edge time constant.	1e-12	s
Tf	numerical	Falling edge time constant.	1e-12	s
Pw	numerical	Pulse width.	0.0	s
Pr	numerical	Repeating period of the pulse train.	0.0	s
User-defined Envelop Parameters				
DLL	string	Name of the dynamically loadable library file that contains the user-defined source.	none	none
Function	string	Function name of the source in the DLL file.	none	none

Examples

```
ENVELOP   ID=gaussian_pulse Type=Gaussian amplitude=1e3 tdelay=0
     t0=5e−12 tao=1e−12
```

CIRCUIT

The **CIRCUIT** command defines a circuit netlist (in SPICE syntax) for mixed-mode simulation.

```
CIRCUIT
  Netlist = <<str>>
```

Parameters

Parameter	Type	Description	Default	Unit
Netlist	string	Path to the netlist file. The netlist should be in SPICE format.	none	none

Boundary Conditions

Boundary and Contact

The **BOUNDARY** statement sets boundary information to representing faces of the device mesh. GENIUS now fully support electrode region (the material of this region should be metal or poly-Si). One should use **CONTACT** statement to specify the electrode type of this region(s).

```
BOUNDARY
    ID=<<str>>
    Type = ( OhmicContact | SchottkyContact | SolderPadContact |
             GateContact | SimpleGateContact |
             InsulatorInterface|
             HeteroJunction | NeumannBoundary )

  Ohmic Contact
    [ Res=<<num>> ] [ Cap=<<num>> ] [ Ind=<<num>> ]
    [ Heat.Transfer=<<num>> ] [ Ext.Temp=<<num>> ]

  Schottky Contact
    [ WorkFunction=<<num>> ]
    [ Res=<<num>> ] [ Cap=<<num>> ] [ Ind=<<num>> ]
    [ Heat.Transfer=<<num>> ] [ Ext.Temp=<<num>> ]

  Gate Contact
    [ WorkFunction=<<num >>]
    [ Res=<<num>> ] [ Cap=<<num>> ] [ Ind=<<num>> ]
    [ Heat.Transfer=<<num>> ] [ Ext.Temp=<<num>> ]

  SolderPad Contact
    [ Res=<<num>> ] [ Cap=<<num>> ] [ Ind=<<num>> ]
    [ Heat.Transfer=<<num>> ] [ Ext.Temp=<<num>> ]
  Simple Gate Contact
    [ WorkFunction=<<num >>] [ QF=<<num>> ]
    [ Res=<<num>> ] [ Cap=<<num>> ] [ Ind=<<num>> ]
    [ Thickness=<<num>> ] [ eps=<<num>> ]
    [ Heat.Transfer=<<num>> ] [ Ext.Temp=<<num>> ]

  Insulator Interface
    [ QF=<<num>> ]

  HeteroJunction
    [ QF=<<num>> ]

  NeumannBoudary
    [ Heat.Transfer=<<num>> ] [ Ext.Temp=<<num>> ]

Contact
    ID=<<str>> Type = (OhmicContact|SchottkyContact|
                      GateContact|FloatMetal)

  Ohmic Contact
    [ Res=<<num>> ] [ Cap=<<num>> ] [ Ind=<<num>> ]
    [ Heat.Transfer=<<num>> ] [ Ext.Temp=<<num>> ]

  SchottkyContact
    [ WorkFunction=<<num>> ]
    [ Res=<<num>> ] [ Cap=<<num>> ] [ Ind=<<num>> ]
    [ Heat.Transfer=<<num>> ] [ Ext.Temp=<<num>> ]

  Gate Contact
    [ WorkFunction=<<num >>]
    [ Res=<<num>> ] [ Cap=<<num>> ] [ Ind=<<num>> ]
    [ Heat.Transfer=<<num>> ] [ Ext.Temp=<<num>> ]

  Floating Metal
    [ QF=<<num>> ]
```

Parameters

Parameter	**Type**	**Description**	**Default**	**Unit**
Type	enum	This parameter declares which type of boundary condition is defined here.	none	none
ID	string	A unique string which identifies the corresponding face.	none	none
Res	numerical	The lumped resistance for the electrode.	0.0	Ω
Cap	numerical	The lumped capacitance for the electrode.	0.0	F
Ind	numerical	The lumped inductance for the electrode.	0.0	H

WorkFunction	numerical	The workfunction of the Schottky contact or gate material.	0.0	eV
QF	numerical	For InsulatorContact and InsulatorInterface bc: The surface charge density of semiconductor-insulator interface.	0.0	cm^{-2}
QF	numerical	For Heterojunction bc: The surface charge density of heterojunction.	0.0	cm^{-2}
QF	numerical	For FloatMetal bc: The free charge per micron in Z dimension.	0.0	$C \times \mu\mathrm{m}^{-1}$
Thickness	numerical	The thickness of SiO2 layer.	2e-7	cm
eps	numerical	The relative permittivity of SiO2 layer.	3.9	none
Heat.Transfer	numerical	The thermal conductance of the surface.	see below	W $\times$ cm^{-2}
Ext.Temp	numerical	The external temperature at the surface.	see below	K

Description

Five "electrode" boundary conditions are supported by GENIUS. The names are ended with **Contact**. The **OhmicContact** and **SchottkyContact** electrodes have current flow in both steady state and transient situations. While **GateContact** and **SimpleGateContact**(a simplified MOSFET Gate boundary condition) only have displacement current in transient situation. **SolderPadContact** also has current flow in steady state and transient situations. The **OhmicContact**, **SchottkyContact** and **SimpleGateContact** are boundaries on surface of semiconductor region. **GateContact** should be the surface of insulator region and **SolderPadContact** is the surface of a metal region.

Genius supports five interfaces which can be set automatically: semiconductor-insulator interface (**InsulatorInterface**), semiconductor-electrode interface(set to **OhmicContract** as default), interface between different semiconductor material (**Heterojunction**) and interface between same semiconductor material (**Homojunction**). These boundaries can be set automatically by Genius if user didn't set them explicitly. However, the electrode-insulator interface, may have several situations: Gate to Oxide interface, **FloatMetal** to Oxide interface or Source/Drain electrode to Oxide interface. As a result, this interface can only be set correctly when electrode type is known. Please refer to the following **CONTACT** statement.

GENIUS can build region with metal or poly-Si material to form an electrode. Which means, i.e. for **OhmicContact** bc, one can simply specify a segment as Ohmic bc or build an electrode region as Ohmic electrode. Since Version 0.45.03, GENIUS considers electrode region, semiconductor region and insulator region during calculation. As a result, GENIUS added **CONTACT** statement for fast boundaries specification of electrode region. At present, GENIUS support electrode with the type of Ohmic, Schottky, Gate and Floating-Metal. All the electrode should be specified explicitly and GENIUS will set corresponding boundaries automatically.

The **ID** parameter of **BOUNDARY** statement is limited to face label. And The **ID** parameter of **CONTACT** statement is limited to region name.

The NeumannBoundary, which is the default boundary type for all the non-interface segments, are set automatically at the outer surface of a region.

The thermal boundary conditions at outer surfaces of the device and the interface with electrode regions are specified by the **Heat.Transfer** and **Ext.Temp** parameters. The default external temperature at the boundary is the external temperature set in the **GLOBAL** command. The default value for heat conductance varies with the types of boundary conditions. For **OhmicContact**, **SchottkyContact** and **Gatecontact**, the default heat conductance is $10^3\mathrm{W} \times cm^{-2}\mathrm{K}^{-1}$. For **NeumannBoundary**, the heat resistance is default to zero.

Example

```
BOUNDARY  Type=SimpleGateContract   ID=SiSiO2  Res=0  Cap=0  Ind=0  \
          Thickness=1e-6  Eps=3.9  WorkFunction=4.7  QF=0

BOUNDARY  Type=InsulatorInterface  ID=IFACE  QF=0

BOUNDARY  Type=GateContract         ID=GATE    Res=0  Cap=0  Ind=0  \
          WorkFunction=4.7

BOUNDARY  Type=NeumannBoundary      ID=WALL    Heat.Transfer=0  \
          EXT.Temp=300

BOUNDARY  Type=SchottkyContract     ID=sgate   Res=0  Cap=0  Ind=0  \
          VBarrier=0.8

BOUNDARY  Type=OhmicContract       ID=OMANODE Res=0  Cap=0  Ind=0

BOUNDARY  Type=SolderPad           ID=CATHODE Res=0  Cap=0  Ind=0
```

ATTACH

The **ATTACH** command is used to add voltage or current sources to the electrode boundary. The statement first clears all the sources connected to the specified electrode and then adds source(s) defined by **VApp** or **IApp** parameter. If two or more sources are attached to the same electrode, the net effect is the superposition of all sources. However, the sources attached to one electrode must have the same type.

```
ATTACH
  Electrode=str { VApp=str | IApp=str } [ { VApp=str | IApp=str
     } ... ]
```

Parameters

Parameter	Type	Description	Default	Unit
Setup Loadable Physical Models				
Electrode	string	Electrode where the external source is attached. Must be a valid ID of a electrical contact specified by the **BOUNDARY** or **CONTACT** commands.	none	none
VApp	string	ID of the voltage source.	none	none
IApp	string	ID of the current source.	none	none

Description

If electrode is attached with voltage source(s), the R, C and L defined by **BOUNDARY** statement will affect later simulation. But solver will ignore R and L when the electrode stimulated by current source(s). Please refer to figure **??**, p. 72.

Several voltage sources or current sources can be attached to the same electrode by multi- **Vapp** or multi-**Iapp** parameter. However, both the voltage source and current source attach to the same electrode is invalid. Only Ohmic, Schottky and SolderPad electrodes can be attached by current source(s). And the positive direction of current is flow into the electrode.

If no source attached explicitly, the electrode is set to be attached to ground (0V VDC).

Figure 2.4: Voltage and current boundary.

Example

```
ATTACH Electrode=Anode Vapp=V5V Vapp=Vs
ATTACH Electrode=Base  Iapp=Ib
```

Fabrication Process

DOPING

The DOPING command selects the mode of doping profile specification. Only analytic doping profile is currently implemented.

```
Doping Analytic
```

Analytic PROFILE

The PROFILE command specifies the doping profile. Uniform and analytic doping profiles are supported.

```
PROFILE [.DOPING]
    [ ID=<<str>> ]
    Type = (Uniform|Analytic|Analytic2|File) Ion = (Donor|
        Acceptor|Custom)

  Uniform Profile Parameters
    N.Peak=<<num>>
    [ x.min=<<num>> ] [ x.max=<<num>> ] [ y.min=<<num>> ] [ y.
        max=<<num>> ]
    [ z.min=<<num>> ] [ z.max=<<num>> ]

  Anaylitic Profile Parameters
    { N.Peak=<<num>> | Dose=<<num>> }
    [ x.min=<<num>> ] [ x.max=<<num>> ] [ y.min=<<num>> ] [ y.
        max=<<num>> ]
    [ z.min=<<num>> ] [ z.max=<<num>> ]
    { Y.Char=<<num>> | Y.Junction=<<num>> } [ Y.ErfC=<<bool>> ]
    { X.Char=<<num>> | XY.Ratio=<<num>> } [ X.ErfC=<<bool>> ]
    { Z.Char=<<num>> | ZY.Ratio=<<num>> } [ Z.ErfC=<<bool>> ]

  Anaylitic2 Profile Parameters
    { N.Peak=<<num>> }
    {
      mask.polygon=<<anum>> |
      { mask.x=<<num>> | mask.y=<<num>> | mask.z=<<num>> }
      [ mask.xmin=<<num>> ] [ mask.xmax=<<num>> ]
      [ mask.ymin=<<num>> ] [ mask.ymax=<<num>> ]
      [ mask.zmin=<<num>> ] [ mask.zmax=<<num>> ]
    }
    [ implant.rmin=<<num>> ] [ implant.rmax=<<num>> ]
    [ implant.theta=<<num>> ] [ implant.phi=<<num>> ]
    [ depth.char=<<num>> ] [ lateral.char=<<num>> ]
    [ resolution=<<num>> ]

  Data File Parameters
    File=<<str>> [ SkipLine=<<num>> ]
    [ Axes= (x|y|xy|xz|yz|xyz) ]
    [ LUnit = (m|cm|um|nm) ]
    [Transform.xx=<<num>>] [Transform.xy=<<num>>] [Transform.xz
        =<<num>>]
    [Transform.yx=<<num>>] [Transform.yy=<<num>>] [Transform.yz
        =<<num>>]
```

```
[Transform.zx=<<num>>] [Transform.zy=<<num>>] [Transform.zz
    =<<num>>]
[Translate.x=<<num>>]  [Translate.y=<<num>>]  [Translate.z
    =<<num>>]
```

Parameters

Parameter	Type	Description	Default	Unit
General Parameters				
Type	enum	Selects the type of doping profile. Doping profile can be **Uniform**, specified by **Analytic** expression, or loaded from a data **File**.	Uniform	none
Ion	enum	Selects the doping species of the profile. Acceptor, donor and custom species are available. Custom species must bear an ID, so that it can be referenced in other commands (e.g. in PMI trapping model and incomplete ionization model).	none	none
ID	string	Identifier of a custom doping profile.	none	none
x.min	numerical	Left boundary of the profile. Synonym: x.left.	0.0	μm
x.max	numerical	Right boundary of the profile. Synonym: x.right.	0.0	μm
y.min	numerical	Top boundary of the profile. Synonym: y.top.	0.0	μm
y.max	numerical	Bottom boundary of the profile. Synonym: y.bottom.	0.0	μm
z.min	numerical	Front boundary of the profile. Synonym: z.front.	0.0	μm
z.max	numerical	Back boundary of the profile. Synonym: z.back.	0.0	μm
Uniform Profile Parameters				
N.Peak	numerical	Doping concentration.	0.0	cm^{-3}
Analytic Profile Parameters				

N.Peak	numerical	Peak doping concentration. In the box defined by **x.min**, **x.max**, **y.min**, **y.max**, **z.min**, **z.max**, the doping is uniformly **N.Peak**. Outside the box, the doping concentration decays according to the analytic function.	0.0	cm^{-3}
Dose	numerical	The dose of the doping concentration, assuming the resulting profile is 1D Gaussian.	0.0	cm^{-2}
Y.Char	numerical	Characteristic length in the y direction.	0.0	μm
Y.Junction	numerical	Target junction depth.	0.0	μm
X.Char	numerical	Characteristic length in the x direction.	0.0	μm
XY.Ratio	numerical	The ratio of characteristic length in the x direction against that in the y direction.	0.0	none
X.Erfc	bool	Selects whether to use erfc instead of Gaussian function in the x direction.	false	none
Z.Char	numerical	Characteristic length in the z direction.	0.0	μm
ZY.Ratio	numerical	The ratio of characteristic length in the z direction against that in the y direction.	0.0	none
Z.Erfc	bool	Selects whether to use erfc instead of Gaussian function in the z direction.	false	none
Analytic2 Profile Parameters				
mask.poly	numerical-array	A numerical array specifies vertex location of the polygon mask. Each vertex takes three number as its coordinates. The vertex should be ordered along the loop of polygon. Polygon should be simple, i.e. no self intersection.	-	$\uMe
mask.x	numerical	Indicate that doping mask plane is the X=num, for rectangle mask.	0.0	$\uMe
mask.y	numerical	Indicate that doping mask plane is the Y=num, for rectangle mask.	0.0	$\uMe

mask.z	numerical	Indicate that doping mask plane is the Z=num, for rectangle mask.	0.0	$\uMe
mask.xmin	numerical	Left boundary of the rectangle mask	0.0	$\uMe
mask.xmax	numerical	Right boundary of the rectangle mask	0.0	$\uMe
mask.ymin	numerical	Top boundary of the rectangle mask	0.0	$\uMe
mask.ymax	numerical	Top boundary of the rectangle mask	0.0	$\uMe
mask.zmin	numerical	Front boundary of the rectangle mask	0.0	$\uMe
mask.zmax	numerical	Back boundary of the rectangle mask	0.0	$\uMe
implant.rmin	numerical	The minimal depth of implant. Doping concentration keeps constant between min and max doping depth.	0.0	$\uMe
implant.rmax	numerical	The maximal depth of implant. Doping concentration keeps constant between min and max doping depth.	0.0	$\uMe
implant.theta	numerical	Defined as angel between +Y direction and implantation direction.	0.0	degre
implant.phi	numerical	Defined as angel between +X direction and the projection of implantation direction onto XZ plane.	0.0	degre
depth.char	numerical	Characteristic length along the implantation direction.	0.1	$\uMe
lateral.char	numerical	Characteristic length lateral to the implantation direction.	0.1	$\uMe
resolution	numerical	The space resolution factor, minimal sapce equals to **lateral.char** divide by this factor.	4.0	none
Doping Data File Parameters				
File	string	Name of the doping concentration data file.	none	none

Transform	numerical	The transformation matrix applied to the coordinates imported from the input file. The final coordinates are obtained from **Transform***location + **Translate**.	I	none
Translate	enum	The translation vector applied to the input coordinates.	0	none
SkipLine	integer	Skip the first **SkipLine** number of lines in the data file.	0	none
LUnit	enum	Length unit used for node coordinates in the data file.	**um**	none

Analytic Profile Calculation

When the **Type** parameter is set to **Analytic**, the doping profile is calculated with the expression

$$N(x, y) = \mathbf{N.PEAK} \cdot f_x(x) \cdot f_y(y) \cdot f_z(z), \tag{2.4}$$

where **N.PEAK** is the peak doping concentration, and the functions $f_x(x)$, $f_y(y)$ and $f_z(z)$ describes the variation of doping concentration in the x-, y- and z-directions, respectively.

Two options are currently available for the variation functions, namely the Gauss profile function and the Erfc profile function. One can choose the profile function for the x-, y- and z-directions independently. When the **X.Erfc**, **Y.Erfc** or **Z.Erfc** parameter is enabled, the Erfc function is used in that direction.

The Gauss profile function is defined as

$$f_D(x, y) = \exp\left[-\left(\frac{y - \mathbf{D.MIN}}{\mathbf{D.CHAR}}\right)^2\right] y < \mathbf{D.MIN} 1 \mathbf{D.MIN} \le y \le \mathbf{D.MAX} \exp \Bigg[- \Bigg(\tag{2.5}$$

where $\mathbf{D} = \mathbf{X}, \mathbf{Y}, \mathbf{Z}$.

The Erfc profile function is defined as

$$N(x, y) = \frac{erfc\left(\frac{y - \mathbf{D.MAX}}{\mathbf{D.CHAR}}\right) - erfc\left(\frac{y - \mathbf{D.MIN}}{\mathbf{D.CHAR}}\right)}{2}. \tag{2.6}$$

If the **XY.Ratio** parameter is specified instead of the **X.Char** parameter, **X.Char** is calculated with $\mathbf{X.Char} = \mathbf{XY.Ratio} \times \mathbf{Y.Char}$. Similar calculation applies to **ZY.Ratio** and **Z.Char**.

Implant Dose

The **Dose** parameter can be used instead of the **N.Peak** parameter to specify the dose of impurities per unit area. In this case, the peak concentration is calculated with

$$N.PEAK = \frac{\mathbf{DOSE}}{\sqrt{\pi}\mathbf{Y.CHAR}}. \tag{2.7}$$

Analytic2 Profile Calculation

When the **Type** parameter is set to **Analytic2**, the doping profile is calculated with a given mask and the implant parameters. A planer rectangle mask or polygon mask should be defined as the window of implantation. The implantation is defined by incident angle given by **implant.theta** and **implant.phi** and ranges given by **implant.rmin** and **implant.rmax**, which relative to the mask plane. The doping concentration equals to **N.Peak** within the range between **implant.rmin** and **implant.rmax**, and variated as Gauss profile function with parameter **depth.char** along the doping line outside the range. For lateral distribution, a Gauss profile with parameter **lateral.char** is assumed. The Genius use a ray shooting method to calculate the final doping concentration. The accurate is depended on density of doping lines which controlled by parameter**resolution**.

Custom Doping Profile

When a custom dopant species is desired, the user should set **Ion** to **Custom**, and supply an identifier **ID**.

Field Data File

The field data file must be in plain text, with numerical values separated by white spaces. For 3D doping profile, each line contains four values (x-, y-, and z-coordinate and the doping concentration). For 2D doping profile, the z-coordinate is not present, and each line contains three values. The unit of the coordinates is specified by **LUnit**, and the unit of the concentration is cm^{-3}. Positive doping concentration indicates Donor ion, while negative doping concentration indicates Acceptor ion.

Example

```
Profile.Doping    Type=Analytic  Ion=Acceptor  \
                  N.Peak=1e18  Y.Min=0.0  Y.Max=0.1  \
                  Y.Char=0.05  XY.Ratio=0.8

Profile.Doping    Type=Analytic2  Ion=Acceptor  N.PEAK=1E19  \
                  Mask.poly=[1,0,2,3,0,3,5,0,1,5,0,5,1,0,4]  \
                  Implant.rmin=0.0  Implant.rmax=0.3  Implant.theta
                     =0

Profile.Doping    Type=Uniform  Ion=Acceptor  N.Peak=1e16
Profile.Doping   Type=File  File="halo.txt"
```

MOLE

The **MOLE** command specifies the mole fraction profile of compound materials. For binary compound material AxB1-x, the mole fraction profile x is needed. For tertiary compound material AxByC1-x-y, both x and y profile are needed.

```
MOLE
    Region=str
    [ x.min=<<num>> ] [ x.max=<<num>> ] [ y.min=<<num>> ] [ y.
       max=<<num>> ]
    [ z.min=<<num>> ] [ z.max=<<num>> ]

  Binary Compound Material
    X.Mole=<<num>> ( X.Mole.Slope=<<num>> | X.Mole.End=<<num>> )
    [ X.Mole.Grad = (X.Linear|Y.Linear|Z.Linear) ]

  Tertiary Compound Material
    X.Mole=<<num>> ( X.Mole.Slope=<<num>> | X.Mole.End=<<num>> )
    [ X.Mole.Grad = (X.Linear|Y.Linear|Z.Linear) ]
    Y.Mole=<<num>> ( Y.Mole.Slope=<<num>> | Y.Mole.End=<<num>> )
    [ X.Mole.Grad = (X.Linear|Y.Linear|Z.Linear) ]
```

Parameters

Parameter	Type	Description	Default	Unit
General Parameters				
Region	string	Identifier of the compound material region.	none	none
x.min	numerical	Left boundary of the profile. Synonym: x.left.	0.0	μm
x.max	numerical	Right boundary of the profile. Synonym: x.right.	0.0	μm
y.min	numerical	Top boundary of the profile. Synonym: y.top.	0.0	μm
y.max	numerical	Bottom boundary of the profile. Synonym: y.bottom.	0.0	μm
z.min	numerical	Front boundary of the profile. Synonym: z.front.	0.0	μm
z.max	numerical	Back boundary of the profile. Synonym: z.back.	0.0	μm
Binary Compound				
X.Mole	numerical	Starting mole fraction x of compound material AxB1-x.	0.0	none

X.Mole.Slope	numerical	The slope of mole fraction x.	0.0	μm
X.Mole.End	numerical	The mole fraction x at the end of the profile.	0.0	none
X.Mole.Grad	enum	The direction of mole fraction gradient.	**Y.Linear**	none
Tertiary Compound				
X.Mole	numerical	Starting mole fraction x of compound material AxByC1-x-y.	0.0	none
X.Mole.Slope	numerical	The slope of mole fraction x.	0.0	μm
X.Mole.End	numerical	The mole fraction x at the end of the profile.	0.0	none
X.Mole.Grad	enum	The direction of mole fraction gradient.	**Y.Linear**	none
Y.Mole	numerical	Starting mole fraction y.	0.0	μm
Y.Mole.Slope	numerical	The slope of mole fraction y.	0.0	none
Y.Mole.End	numerical	The mole fraction y at the end of the profile.	0.0	none
Y.Mole.Grad	enum	The direction of mole fraction gradient.	**Y.Linear**	none

Description

Only linear mole fraction profile with its gradient aligned with one of the three Cartesian axes is currently supported. There are two alternative ways to specify the expression of the profile. One can either specify the slope explicitly with **X.Mole.Slope** and the mole fraction at the starting end with **X.Mole**. Starting end is defined as the one with smaller coordinate value in the direction of **X.Mole.Grad**. Alternatively, one can set the mole fraction at both ends of the region with **X.Mole** and **X.Mole.End**, and let GENIUS calculate the gradient.

Example

```
MOLE    region=SiGe  X.Mole=0.4  X.Mole.End=0.7  \
        X.Mole.Grad=Y.Linear

MOLE    region=SiGe  X.Mole=0.4  X.Mole.Slope=0.1  \
        Y.Min=0  Y.Max=0.3  X.Mole.Grad=Y.Linear
```

Physical Models

PMI

In Genius, the physical models of each material is loaded to the simulation system through the physical model interface (PMI). The models can be selected through the PMI command. The model parameters are also specified here. The general syntax of the PMI command is

```
PMI
  REGION=str
  [ Type= (Basic|Band|Mobility|Impact|Thermal|Optical|Trap) ]
  [ Model=<<str>> ] [print=<<int>>]
  [ real<<<str>>>=<<num>> [ real<<<str>>>=<<num>> [...] ] ]
  [ string<<<str>>>=<<str>> [ string<<<str>>>=<<str>> [...] ] ]
```

Parameters

Parameter	Type	Description	Default	Unit
Setup Loadable Physical Models				
Region	string	The region that the loaded model would apply to. A valid region name must be supplied, otherwise the simulation will abort.	none	none

Type	enum	The type of model to be loaded. Seven types are currently supported in Genius, though some of them are not applicable to all material regions. • A **Basic** model specifies the basic physical parameters such as permittivity and electron affinity, etc. • A **Band** model specifies parameters related to the band structure of semiconductors, which include energy bandgap, effective density of states, effective masses, etc. It also calculates rates of some fundamental processes such as carrier recombination rates. **Band** models are applicable to semiconductor regions only. • A **Mobility** model calculates the mobility of carriers, taking into consideration the effect of temperature, doping, electric field and adjacent interfaces, etc. A range of mobility models are provided in Genius, which will be discussed in more detail in Mobility Models, p. 84. Mobility models apply to semiconductor regions only. • An **Impact** model calculates the carrier generation rates due to the impact ionization process. A later section will describe impact ionization models in detail. **Impact** models apply to semiconductor regions only. • An **Thermal** model	Mobility	none

Model	string	Name of the physical models to be loaded. The list of available models for each material and each type of models is summarized in later sections.	Default	none
Print	integer	If greater than 0, print the current values of all the material parameters.	0	none
real < **str** >	numerical	Additional custom numerical parameters accepted by the loaded model. The name of the parameter is specified by **str**.	none	unknown
string < **str** >	string	Additional custom string parameters accepted by the loaded model. The name of the parameter is specified by **str**.	none	none

Description

The core program of Genius access the numerous material-specific models via the Physical Model Interface (PMI).

In Genius, each supported material has a corresponding dynamic loadable library (DLL) file located in the FilenameType $(GENIU_DIR)/lib/ directory. Each DLL file contains all models applicable to that material. For instance, all physical models for Si material are stored in FilenameType $(GENIUS_DIR)/lib/libSi.so.

For each type of models, a default model will be loaded in the absence of a **PMI** command. When the user supplied one or more **PMI** commands, Genius sequentially loads the specified model and assign it to the specified region. The user can optionally supply a list of custom numerical and string parameters to the PMI model. For instance, the following command loads the trap model for Silicon region, and setup the parameters of the trap model.

```
PMI   Region=Silicon  Type=Trap  string<Type>=Bulk  \
      string<Profile>=dopingA  string<ChargeType>=Acceptor  \
      double<Energy>=0.1
```

Each PMI model has its own list of accepted parameters, and the user should consult the description of the individual PMI models for details.

If the model specified by a **PMI** command has already been loaded, the previously loaded one is *not* replaced. The latter **PMI** command are useful in adjusting the custom parameters of the PMI model.

Custom PMI Model

Advanced users of Genius can compile and produce dynamic libraries that contain custom PMI models. The easiest way of writing custom PMI model is to extend from the source code of existing models. See [[TODO]] for a tutorial guide and detailed references.

Mobility Models

The mobility model can be selected in the **Model** parameter of the PMI command:

```
PMI
    REGION=<<str>> Type=Trap [ Model=<<str>> ]
    [ <<param>>=<<val>> [ <<param>>=<<val>> [ ... ] ] ]
  %
```

The variety of available mobility models and the default model for each material is summarized in table **??**, p. 84. Each model has a set of its own parameters, which can be adjusted with the PMI command. The description of each mobility model and the corresponding parameters are detailed in Mobility Models, p. 19.

Table 2.21: Summary of Available PMI Mobility Models

Material	Models	Default
3C-SiC	**Analytic**	**Analytic**
4H-SiC	**Masetti**	**Masetti**
AlGaAs	**Analytic**	**Analytic**
GaAs	**Analytic**, **Hypertang**	**Analytic**
Ge	**Analytic**	**Analytic**
HgCdTe	**Analytic**	**Analytic**
InAs	**Analytic**	**Analytic**
InGaAs	**Analytic**	**Analytic**
InN	**Analytic**	**Analytic**
InP	**Analytic**	**Analytic**
InSb	**Analytic**	**Analytic**
S-SiO2	**Constant**	**Constant**
Si	**Constant**, **Analytic**, **Philips**, **HP**, **Lucent**, **Lombardi**	**Analytic**
SiGe	**Constant**, **Analytic**, **Philips**, **HP**, **Lucent**, **Lombardi**	**Analytic**

The following example shows how one can load the HP mobility model and adjust the default parameter.

```
PMI   Region=Silicon  Type=Mobility  Model=HP \
      double<MUN.MAX>=1200  double<MUP.MAX>=400
```

Impact Ionization Models

The mobility model can be selected in the **Model** parameter of the PMI command:

```
PMI
```

```
REGION=<<str>> Type=Impact [ Model=<<str>> ]
[ <<param>>=<<val>> [ <<param>>=<<val>> [ ... ] ] ]
```

For silicon and many other materials, the **Default** avalanche generation model is the Selberherr model. The models for various materials are summarized in table **??**, p. 85.

Table 2.22: Summary of Available PMI Impact Ionization Models

Material	Models	Default
3C-SiC	**Selberherr**	**Selberherr**
4H-SiC	**vanOverstraetendeMan**	**vanOverstraetendeMan**
AlGaAs	**Selberherr**	**Selberherr**
GaAs	**Selberherr**	**Selberherr**
Ge	**Selberherr**	**Selberherr**
HgCdTe	-	-
InAs	**vanOverstraetendeMan**	**vanOverstraetendeMan**
InGaAs	**Selberherr**	-
InN	**vanOverstraetendeMan**	**vanOverstraetendeMan**
InP	**Selberherr**	**Selberherr**
InSb	**vanOverstraetendeMan**	**vanOverstraetendeMan**
S-SiO2	-	-
Si	**Selberherr, Valdinoci**	**Selberherr**
SiGe	**Selberherr**	**Selberherr**

Trap Model

Although the charge trapping model is automatically loaded, the user needs to setup trap parameters with **PMI** command for it effect. The syntax and parameters are as follows.

```
PMI
    REGION=<<str>> Type=Trap Model=Default
    [string<Type> = (Bulk|Interface)]
    [string<ChargeType> = (Acceptor|Donor)]
    [real<Energy>=<<num>>] [real<SigmaN>=<<num>>] [real<SigmaP
        >=<<num>>]

  Bulk Trap
    { string<Profile>=<<str>> [ real<Prefactor>=<<num>> ]  |

  Interface Trap
    string<Interface>=<<str>> real<IF.Density>=<<num>> }
```

Parameter	Type	Description	Default	Unit
Common Trap Parameters				

Type	enum	Selects whether the current command specifies **Bulk** or **Interface** traps.	**Bulk**	none
ChargeType	enum	Selects whether the traps is **Acceptor** or **Donor**.	**Acceptor**	none
Energy	numerical	Energy of the trap level relative to the intrinsic Fermi level of the material.	0	eV
SigmaN	numerical	Capture cross-section for electrons.	4e16	cm^{-2}
SigmaP	numerical	Capture cross-section for holes.	4e16	cm^{-2}
Bulk Trap Parameters				
Profile	string	The concentration of bulk traps is derived from a custom doping profile defined by a **PROFILE** command. ID of the custom profile should be specified here.	none	none
Prefactor	numerical	The concentration of the doping profile specified by **Profile** is scaled by this **Prefactor**. This is useful in generating a number of traps with the same spatial distribution but different trap energies.	1.0	none
Interface Trap Parameters				
Interface	string	Interface traps are attached to the insulator/semiconductor interface with the ID specified here. When an semiconductor region and an insulator region share a common face, an interface is automatically generated. The ID of the interface is in the format of $\mathbf{RegionName1_{t}o_{R}egionName2}$. Alternatively, one can use a **Face** command to define an interface with a user-specified ID.	none	none
IF.Density	numerical	The interface density of traps.	0.0	cm^{-2}

Incomplete Ionization Model

User can setup incomplete ionization parameters with **PMI** command. Please note, for active this model, use also have to set **IncompleteIonization** to true in the **MODEL** statement. Also, it is recommend to use Fermi statistics with IncompleteIonization. The syntax and parameters are as follows.

```
PMI
    REGION=<<str>> Type=band
    [string<species>=<<str>>] [int<ion>=(+1|-1)]
    [real<EB0>=<<num>>] [real<GB>=<<num>>]
    [real<alpha>=<<num>>] [real<beta>=<<num>>] [real<gamma>=<<
        num>>]
 %
```

Parameter	Type	Description	Default	Unit
Common Trap Parameters				
species	string	Specify profile species. The string parameter is the id of profile with Custom ion type.	**none**	none
ion	integer	Selects whether the species is **Acceptor**(-1) or **Donor**(+1).	**0**	none
EB0	numerical	The constant activation energy.	0	eV
GB	numerical	The band degeneracy factor	none	none
alpha	numerical	The prefactor for the doping dependent term used in the calculation of the band ionization energy.	none	eV cm
beta	numerical	The prefactor the temperature dependent term used in the calculation of the band ionization energy.	none	none
gamma	numerical	The exponent of temperature used in the calculation of the band ionization energy.	none	none

MODEL

The MODEL command sets the switches that controls the behavior of the physical equations.

```
MODEL
  Region=str
  [ Fermi=bool ] [IncompleteIonization=bool]
  { H.Mob=bool [ Mob.Force = (EJ|ESimple|EQF) ] }
  { ImpactIonization = (Local|No)
    [ II.Force = (EdotJ|ESide|EVector|GradQF) ] }
  [ BBT = (Local|No) ]
  [ Optical.Gen = (true|false) ]
  [ Particle.Gen = (true|false) ]
  [ EB.Level= (None|Tl|Te|Th|TeTh|TeTl|ThTl|ALL) ]
```

Parameters

Parameter	Type	Description	Default	Unit
Region	string	Select models in the specified region.	none	none
Fermi	boolean	Turns on Fermi-Dirac carrier statistics.	false	none
IncompleteIonization				
H.Mob	boolean	Synonym to **HighFieldMobility**. Activate high-field mobility calculation. When **H.Mob** is true, parallel and transverse fields are calculated and passed to the mobility model, so that high-field effects on carrier mobility are taken into account.	true	none
ESurface	boolean	Use effective surface electric field for carrier mobility calculation at insulator-semiconductor interface.	true	none
Mob.Force	enum	Synonym to **Mobility.Force**. When **H.Mob** is enabled, this parameter selects the driving field used in high-field mobility calculation. • When **EJ** is selected, the dot product between electric field and current density ($\vec{E} \cdot \vec{J}/\lvert\vec{J}\rvert$) is used as the parallel field. The cross product ($\vec{E} \times \vec{J}/\lvert\vec{J}\rvert$) is used as the transverse field. • When **ESimple** is selected, the electric field $\vec{E}$ as the parallel field. • When **EQF** is selected, the gradient of the quasi-fermi level ($\nabla\phi_n$ and $\nabla\phi_p$) as the parallel field. The transverse field is calculated with $\vec{E} \times \vec{J}/\lvert\vec{J}\rvert$, as in **EJ** method.	ESimple	none

ImpactIonization	enum	Activates the impact ionization (II) generation, and selects whether the II generation should be calculated with the local model or the non-local model. Only the local model is currently implemented.	No	none
II.Force	enum	When **ImpactIonization** is enabled, this parameter selects the driving field used in impact ionization coefficient calculation. • When **EdotJ** is selected, the dot product between electric field and current density ($\vec{E} \cdot \vec{J}/\lvert\vec{J}\rvert$) is used as the driving field. • When **EVector** is selected, the magnitude of the electric field $\vec{E}$ is used as the driving field. • When **ESide** is selected, the E-field component along the edge of the mesh element is used as the driving field. • When **GradQf** is selected, the gradient of the quasi-fermi level ($\nabla\phi_n$ and $\nabla\phi_p$) as the driving field.	No	none
BBT	enum	Activates the Band-to-Band Tunneling (BBT) generation, and selects whether the BBT generation should be calculated with the local model or the non-local model. Only the local model is currently implemented.	No	none

EB.Level	enum	When the Energy-Balance equation solver is selected in the **METHOD** command, this parameter selectively enables the equations for lattice temperature, electron temperature and hole temperature.	None	none

Example

```
MODEL  Region=emitter  Fermi=true
MODEL  Region=silicon  H.Mob=true  Mob.Force=EQF
MODEL  Region=silicon  EB.Level=TeTl
```

Numerical Solution

METHOD

The **METHOD** command specifies the parameters of the numerical solvers.

```
METHOD

  Equation Selection
    [ Type = (Poisson|DDML1|DDML1Mix|DDML2|DDML2Mix|
              EBML3|EBML3Mix|HalfImplicit|Stress) ]

  Numeric Solver Selection
    [ NS = (Newton|LineSearch|TrustRegion) ]
    [ LS = (CGS|BICG|BCGS|GMRES|TFQMR|LU|UMFPACK|SuperLU|
            MUMPS|SuperLU_Dist) ]
    [ PC = (Identity|Jacobian|BJacobian|ASM|ILU) ]
    [ Damping = (No|Potential|BankRose) ]
    [ Truncation = (Always|Boundary|No) ]
    [ Halfimplicit.let = num ]

  Convergence Criteria
    [ MaxIteration=int ]
    [ Relative.Tol=num ] [ Poisson.Tol=num ]
    [ Elec.Continuity.Tol=num ] [ Hole.Continuity.Tol=num ]
    [ Elec.Energy.Tol=num ] [ Hole.Energy.Tol=num ]
    [ Latt.Temp.Tol=num ] [Electrode.Tol=num]
    [ Toler.Relax=num ] [ QNFactor=num ] [ QPFactor=num ]
```

Parameters

Parameter	Type	Description	Default	Uni

Equation Selection

Type	enum	Selects the set of partial differential equations to be solved. • **Poisson** selects the Poisson's equation solver. • **DDML1** selects the basic level-one drift-diffusion equation solver. • **DDML1Mix** selects the device-circuit mixed-mode solver with the level-one drift-diffusion equation solver. • **DDML2** selects the level-two drift-diffusion equation solver, which includes the lattice temperature variation in device. • **DDML2Mix** selects the device-circuit mixed-mode solver with the level-two drift-diffusion equation solver. • **EBML3** selects the energy-balance equation solver which includes carrier temperature variation in device. Users can selectively activate the equations for electron temperature, hole temperature and lattice temperature with the **EB.Level** parameter in the **MODEL** command. • **EBML3Mix** selects the device-circuit mixed-mode solver with the energy-balance equation solver. • **HalfImplicit** selects the fast half implicit solver. This solver only support transient and OP simulation. And it runs typically 5x faster than DDM solver.	**DDML1**	non

Numerical Solver Selection				
NS	enum	Selects the Newton nonlinear solver algorithm. • **Newton** (synonym: **Basic**) selects the basic nonlinear update algorithm with full Newton steps. • **LineSearch** selects the cubic line-search update algorithm. • **TrustRegion** selects the trust-region update algorithm.	**LineSearch**	non
LS	enum	Selects the linear solver algorithm.	**BCGS**	non
PC	enum	Selects the linear preconditioner algorithm.	**ILU**	non
Damping	enum	Selects the nonlinear update damping scheme.	**Potential**	non
Truncation	enum	Selects the element truncation strategy used in simulation.	**Always**	non
Halfimplicit.let	numerical	Linearize error threshold of half implicit method. A larger threshold will speedup the simulation, but introduce more error.	1.0	Vt
Convergence Criteria of Nonlinear Solver				
MaxIteration	integer	Sets the number of maximum nonlinear iterations before the solver reports failure.	30	non
Relative.Tol	numerical	Sets the relative update tolerance.	1e-5	non
Poisson.Tol	numerical	Sets the absolute tolerance for the residue norm of Poisson's equation.	1e-26	non
Elec.C.Tol	numerical	Sets the absolute tolerance for the residue norm of the electron continuity equation. Synonym: **Elec.Continuity.Tol**.	5e-18	non
Hole.C.Tol	numerical	Sets the absolute tolerance for the residue norm of the hole continuity equation. Synonym: **Hole.Continuity.Tol**.	5e-18	non

Elec.E.Tol	numerical	Sets the absolute tolerance for the residue norm of the electron-energy balance equation. Only applicable when the EBM solver is used and the electron energy-balance equation is activated. Synonym: Elec.Energy.Tol.	1e-18	non
Hole.E.Tol	numerical	Sets the absolute tolerance for the residue norm of the hole energy balance equation. Only applicable when the EBM solver is used and the hole energy-balance equation is activated. Synonym: Hole.Energy.Tol.	1e-18	non
Latt.Temp.Tol	numerical	Sets the absolute tolerance for the residue norm of the lattice heat equation. Applicable when the DDML2 solver is selected; or when the EBM solver is used and the lattice heat equation is activated.	1e-11	non
Electrode.Tol	numerical	Sets the absolute tolerance for the residue norm of at the electrode boundaries.	1e-9	non
Toler.Relax	numerical	Sets the tolerance relaxation factor for convergence on relative tolerance criteria. When the relative error is below Relative.Tol, the solver checks the absolute residue norms of all equations before it reports relative convergence. The absolute residue norms must be below Toler.Relax*aTol, where aTol is the various absolute tolerance described above.	1e4	non
Convergence Criteria of Linear Solver				
KSP.RTol	numerical	Relative tolerance of convergence criterion when ksp methods are usedrelative tolerance of convergence criterion when ksp methods are used.	1e-8	non

$\mathbf{KSP.ATol}$	numerical	Absolute tolerance of convergence criterion when ksp methods are used.	1e-20	non
$\mathbf{KSP.ATol.FNorm}$	numerical	Absolute tolerance convergence criterion, when the residue norm of ksp iteration less than $\mathbf{ksp.atol.fnorm}* \lVert \mathbf{F}(\mathbf{x}) \rVert$.	1e-7	non

Nonlinear Solver Algorithms

GENIUS solves the coupled Poisson-Drift-Diffusion equations in semiconductor devices with Newton's method. The system of nonlinear equations is written as $\mathbf{F}(\mathbf{x}) = 0$, where $\mathbf{F}$ is the nonlinear function and $\mathbf{x}$ is the solution. Starting from an initial guess to the solution, one can solve the nonlinear equation iteratively. The update to the solution in every iteration is computed from the following linear system of equations,

$$\mathbf{J}\left(\mathbf{x}\right) \cdot \mathbf{p} = -\mathbf{F}\left(\mathbf{x}\right) \tag{2.8}$$

where $\mathbf{p}$ is the computed update, and $\mathbf{J}$ is the Jacobian of $\mathbf{F}$. In the basic Newton algorithm, the new solution $\mathbf{x}'$ is

$$\mathbf{x}' = \mathbf{x} + \mathbf{p}. \tag{2.9}$$

This iterative process continues until the convergence criteria is met. Apart from the basic algorithm described above, two variants of Newton solution update algorithms are provided. The cubic linesearch algorithm and the Trust-Region update algorithms can be selected with the $\mathbf{NS}$ parameter.

Update damping algorithms can be used with Newton methods to improve stability of the iterative process. All updates are damped so that the carrier densities are positive. When the $\mathbf{Basic}$ Newton algorithm is selected, one can turn on the Potential damping or the Bank-Rose damping, which sometimes improve convergence.

Nonlinear Convergence Criteria

The nonlinear solver checks for convergence after Newton iteration. The following sets of criteria are checked in sequence, if any one set of criteria is met, the solution is determined to have reached convergence.

Absolute Residue Norm Convergence

The residue of each device equation is evaluated. If the norm of each residue is less than the respective absolute tolerance, convergence is reached.

Relative Residue Norm Convergence

The residue of all device equations is evaluated, and its norm compared against the residue norm before the nonlinear iterations. If the ratio between the two norm is less than the relative tolerance, the absolute residue norm criteria is further checked with the absolute tolerance relaxed by the factor $\mathbf{Toler.Relax}$. Convergence is reached if both tests passes.

Relative Update Norm Convergence The norm of the update in the Newton iteration is evaluated, if this norm is less than the relative tolerance, the absolute residue norm criteria is further checked with the absolute tolerance relaxed by the factor $\mathbf{Toler.Relax}$. Convergence is reached if both tests passes.

Linear Solver Algorithms

equation (**??**), p. 95 can be solved by direct method based on LU factorization. Direct linear solvers usually gives the exact solution of equation (**??**). A few direct linear solvers based on LU decomposition are also provided. The default **LU** solver only supports serial execution. The UMFPACK solver is a fast serial direct solver, while MUMPS supports parallel computation. The serial and parallel version of SuperLU are also available.

Alternatively, the more efficient iterative methods based on Krylov subspace (KSP) theory can be used. However, iterative methods can only give approximate results. Among the choices of linear system solvers, the **CG**, **BCGS**, **GMRES** methods are KSP iterative solvers. A preconditioner is necessary for effective operation of iterative solvers. The **ILU** algorithm usually provides good performance. All the iterative solver and preconditioners listed here support parallel computation. The KSP iteration stops when any of the two criteria is met:

- the NORM change between two iteration less than ksp.rtol.
- residue NORM less than $\max(\mathbf{ksp.atol.fnorm}{*}(\|\mathbf{F}(\mathbf{x})\|), \mathbf{ksp.atol})$.

The iterative linear solver may fail to reach convergence in some cases, which cause the nonlinear solver fail with the error $\mathbf{DIVERGED_LINEAR_SOLVER}$. In this case one may try relaxing the three linear solver tolerance parameters. On the other hand, if the error norm of the nonlinear solver stagnates when a KSP linear solver is in use, it may be caused by inexact linear solution. Tightening the KSP convergence criteria may help.

SOLVE

The **SOLVE** command instructs Genius to solve the semiconductor device equations.

```
SOLVE
  [Type=(Equilibrium|SteadyState|DCSweep|Op|ACSweep|Transient)]
  [Predict=bool]

  DCSweep Simulation Parameters
    { VScan=str  VStart=num  VStep=num  VStop=num
```

```
      [ VStepMax=num ] } |
    { IScan=str IStart=num IStep=num IStop=num
      [ IStepMax=num ] }
    [ Optical.Gen=bool ] [ Optical.modulate=str ]
    [ Particle.Gen=bool ]

  Operator Point Parameters for Mixed-type Simulation
    { [ rampup.steps=int ] [ rampup.vstep=num ]
      [ rampup.istep=num ]
      [ gmin=num ] [ gmin.init=num ]
    }

  Operator Point Parameters for Device Simulation
    { [ pseudotime=bool ] [ pseudotime.step=num ]
      [ pseudotime.iteration=int ]
      [ VStepMax=num ] [ IStepMax=num ]
    }

  Transient Simulation Parameters
    { TS=(ImplicitEuler|BDF1|BDF2|TRBDF2)
      TStart=num TStep=num TStop=num
      [ AutoStep=bool ] [ TStepMax=num ]
      [ TS.rTol=num ]   [ TS.aTol=num ]
      [ VStepMax=num ]  [ IStepMax=num ]
      [ Optical.Gen=bool ] [ Optical.modulate=str ]
      [ Particle.Gen=bool ] [Source.coupled=bool]
      [ tranop=bool ]
    }

  ACSweep Simulation Parameters
    { VScan=str f.start=num f.stop=num
      [ f.multiple=num ]
      [ vac=num ]
    }


  Input/Output Parameters
    [ Hook=str ] [ Out.Prefix=str ]
```

Parameters

Parameter	Type	Description	Default	Un
General Parameters				

Type	enum	Selects the mode of the simulation. • **Equilibrium** sets all external sources to zero, and timestep to infinity, so that the device equations yield equlibrium-state solution. • **SteadyState** sets all external sources to their respective value at time zero, and timestep to infinity. • **DCSweep** scans the external voltage (or current) source at one of the contacts. At each voltage (or current) step, the device equations are solved in steadystate. • **OP** solves the operator point of device or circuit at time zero, some convergence acceleration techniques such as gmin step in circuit simulation and pseudo time method for device can be used. • **ACSweep** applies an small-signal stimulus at one of the contacts, and scans the frequency of the signal. • **Transient** solves the device equation with physical quantities in the device and external sources vary as time progresses.	**Equilibrium**	no

Predict	bool	When **Predict** is enabled, the solver use predictions based on previous solutions as the initial guess to the new solution. When three previous solutions are available, quadratic projection is used. When there are two previous solutions, linear projection is used. When **Predict** is disabled, the solver uses the previous solution as the initial guess. **Predict** is applicable to DC-sweep and transient modes only.	true	nor
Optical.Gen	boolean	Consider optical carrier generation in the simulation. Applicable in DC- and transient-mode simulations only.	false	nor
Optical.modulate	string	Optical wave can be modulated by an envelope, which is specified by **ENVELOP** statement. When this parameter is omitted, constant envelop with amplitude 1.0 will be used. For DC simulation, value at transient time 0 will be used.	none	nor
Particle.Gen	boolean	Consider carriers generated by high-energy particles in the simulation. Applicable in DC- and transient-mode simulations only.	false	nor
Source.coupled	boolean	Indicate that the light absorption or particle generation is coupled with semiconductor device. Thus, Genius will re-calculate carrier generation on each device state.	false	nor
DC Sweep Simulation Parameters				
VScan	string	The electrode on which voltage sweep is applied.	none	nor
VStart	numerical	Start voltage.	none	V

VStep	numerical	Initial voltage step. The actual voltage steps may change when there is convergence problem.	none	V
VStop	numerical	Stop voltage.	none	V
VStepMax	numerical	Maximum voltage step.	**VStep**	V
IScan	string	The electrode on which current sweep is applied.	none	no
IStart	numerical	Start current.	none	A
IStep	numerical	Initial current step. The actual current steps may change when there is convergence problem.	none	A
IStop	numerical	Stop current.	none	A
IStepMax	numerical	Maximum current step.	**IStep**	A
Circuit Operation-Point Simulation Parameters for Mixed Simulation				
rampup.steps	integer	The number of ramp-up steps for the voltage and current sources in the circuit to their DC value.	1	no
rampup.vstep	numerical	The maximum changes of voltage in any ramp up step.	0.25	V
rampup.istep	numerical	The maximum changes of current in any ramp up step.	0.1	A
gmin.init	numerical	The initial value of gmin in gmin step-down. A large value may help convergence for difficult circuits.	1e-12	Ω
gmin	numerical	The target value of gmin step-down. This value should be sufficiently samll to avoid error in circuit simulation.	1e-12	Ω
Pseudo-Time DC Simulation Parameters				
pseudotime	bool	When **pseudotime** is set to true, Genius will use pseudo-time method to calculate the steady-state solution of the device. Currently available in pure device simulation, with **type** = **OP**.	**false**	no
pseudotime.step	numerical	Time step for the pseudo-time method. Small time step will make the system more stable but require more time steps to drive the system to steady state.	$\mathbf{1e-6}$	s

pseudotime.iter	numerical	Maximum number of pseudo-time iteration steps for driving the system to steady state.	**20**	nor
VStepMax	numerical	Maximum voltage update in any pseudo-time step.	**0.1**	V
IStepMax	numerical	Maximum current update in any pseudo-time step.	**0.1**	A
Transient Simulation Parameters				
TS	Enum	Sets the time-discretization scheme. Available choices are BDF1 (aka. Implicit Euler), BDF2 and TRBDF2.	**BDF2**	nor
TStart	numerical	Initial time.	0.0	s
TStep	numerical	Initial time step. The actual time steps is adaptively determined by the solver if AutoStep is enabled.	none	s
TStop	numerical	Stop time.	none	s
TStepMax	numerical	The maximum allowed time step when the solver automatically determines time steps. If TStepMax is zero, there is no upper bound for time steps.	0.0	s
AutoStep	bool	Sets if the solver can adaptively change the time steps.	true	nor
TS.rTol	numerical	Sets the relative truncation error tolerance in time-discretization. When the tolerance criteria is not met, the current time step is rejected, and time step is reduced.	1e-3	nor
TS.aTol	numerical	Sets the absolute truncation error tolerance in time-discretization. When the tolerance criteria is not met, the current time step is rejected, and time step is reduced.	1e-4	nor
VStepMax	numerical	Limit the time step so the maxim change of electrode voltage is less than this value	1.0	V
IStepMax	numerical	Limit the time step so the maxim change of electrode current is less than this value	1.0	A

tranop	bool	Sets if operator point should be computed before transient simulation start. Only valid in Mixed-type Simulation. This option shoule be set to false when **OP** already calculated.	true	no
AC Sweep Simulation				
VScan	string	The electrode on which small AC signal is applied.	none	no
f.start	numerical	Start frequency for AC sweep	1e6	Hz
f.stop	numerical	Stop frequency for AC sweep.	10e9	Hz
f.multiple	numerical	Specifies multiplicative factor for incrementing frequency.	1.1	no
vac	numerical	The magnitude of AC signal.	0.0026	V
Input/Output				
Output.Prefix	string	Sets the prefix to the filename used for output. The internal output routine and the loadable hook functions will append their respective postfix and extensions to form the complete output filename.	none	no

Device DC Sweep Mode

The DC Sweep Mode is selected by setting **type** = **DCSweep**. In this mode, Genius scans the external voltage (or current) source at the contact selected in the **VScan** parameter. At each voltage (or current) step, the device equations are solved in steadystate.

Failure Recovery

When the numerical solver fails to converge at a particular step, Genius recovers to the state of the previous converged step, reduces the step, and attempts to solve the equations again. In **DCSweep** mode, the voltage or current step is recursively halved when the solver diverges. When the solver converges at some reduced step, it tries to progressively double the step it recovers to the initial step (**VStep** or **IStep**).

Device Transient Mode

The Transient Mode is selected by setting **type** = **Transient**. In this mode, external sources (electrical or optical sources) vary as time progresses, and Genius solves the device equation with the time partial-derivative included.

Initial State

An initial device state must be prepared prior to the transient simulation, otherwise the device state at time zero is undetermined. Either a DC Sweep, an Operation Point or a Transient simulation can fulfill this requirement.

Adaptive Time Step

In the $\mathbf{Transient}$ mode, one can specify whether adaptive time step is used with the option $\mathbf{AutoStep}$. When adaptive step is turned off, the initial timestep $\mathbf{TStart}$ is used throughout the simulation. With $\mathbf{AutoStep}$ turned on, the initial time step $\mathbf{TStep}$ is used in the first few steps, and the time-domain truncation error is evaluated. If the truncation error exceeds that specified in $\mathbf{TS.rTol}$ and $\mathbf{TS.aTol}$, the timestep is reduced to 0.9 of the previous value. On the other hand, when the truncation error meets the tolerance criteria, time step is increased by the factor of 1.1, until the maximum timestep $\mathbf{TStepMax}$ is reached.

One can use the $\mathbf{VStepMax}$ and $\mathbf{IStepMax}$ parameters to limit the maximum change in voltage (current) of any voltage (current) source during any time step. When any driving source has a sudden change, the time step will be reduced accordingly.

As in the DC Sweep mode, Genius will recover from convergence failure in the Transient mode, cut the time step to half, and continue solving.

Device Operation Point Mode

The Operation Point (OP) mode is selected by setting $\mathbf{type} = \mathbf{OP}$, and is commonly used to bring the device to a steady-state condition, as the starting state for a DC Sweep or Transient simulation. For better convergence, Genius will ramp up the voltage and current sources from equilibrium (zero) to the specified source values. To limit the maximum change in source values in each ramp-up step, one can set the $\mathbf{VStepMax}$ and $\mathbf{IStepMax}$ parameters.

Pseudo-Time Method

New in 1.7.2. In $\mathbf{OP}$ mode for device simulation, pseudo-time method can be used to better handle ill-conditioned problems, e.g. devices with floating regions, which are traditionally difficult to converge and requires a direct linear solver. With the more numerically stable pseudo-time method, such difficult problems can converge with iterative linear solvers such as $\mathbf{BCGS}$, drastically reducing memory consumption of large-scale problems. In benchmark tests, the pseudo-time method (with BCGS) is modestly faster than traditional steady-state method with the MUMPS direct linear solver.

The flowchart of the pseudo-time method is shown in figure **??**, p. 103, with comparison to the normal steady-state method.

Figure 2.5: Floatchart of the Device Operation Point

Under the pseudo-time scheme, an artificial time-derivative term is added to the electron and hole continuity equations in semiconductor regions ($\partial n/\partial t$ and $\partial p/\partial t$) . At the same time, an artificial capacitance ($\partial V/\partial t$) is added between any resistive metal region and the ground. The system of equations is thus solved in transient mode with the nominal time step specified by the $\mathbf{pseudotime.step}$ parameter.

The transient simulation consists of two stages, as shown in figure **??**, p. 103. In the first stage, voltage and current sources are ramped up. After the target voltage/current are set, Genius solves the equations for a few further steps in the second stage. As time progresses, the device settles down to its steady state, and the change of internal variables (e.g. V, n, and p) between subsequent steps diminishes. After this change drops below the error threshold, the device is deemed to have reached steady-state, and the pseudo-time algorithm completes.

As in the transient and DC sweep modes, in the OP mode, Genius will recover from convergence failure, cut back ramp step or time step, and continue solving.

Device Small-Signal AC Sweep Mode

The AC Sweep Mode is selected by setting $\mathbf{type} = \mathbf{ACSweep}$. In this mode, a small-signal sinusoidal sources is applied on the contact specified by the $\mathbf{VScan}$ parameter, and the AC response of the device is simulated. Before the AC simulation, A steady-state device state must be prepared by means of OP or DC Sweep simulation.

Device/Circuit Mixed Operation Point Mode

When a circuit netlist is present, the device/circuit mixed OP Mode is selected by setting $\mathbf{type} = \mathbf{OP}$.

Device/Circuit Mixed DC Sweep Mode

When a circuit netlist is present, the device/circuit mixed DC Sweep Mode is selected by setting $\mathbf{type} = \mathbf{DCSweep}$.

Device/Circuit Mixed Transient Mode

When a circuit netlist is present, the device/circuit mixed Transient Mode is selected by setting $\mathbf{type} = \mathbf{Transient}$.

Example

```
SOLVE   Type=Equilibrium

SOLVE   Type=DCSWEEP Vscan=Anode  Vstart=0.0  Vstep=0.05  \
        Vstop=0.8  out.prefix=diode_iv

SOLVE   Type=Transient  TStart=0  TStep=1e-12  TStepMax=1e-6  \
        TStop=100e-6
```

EMFEM2D

The **EMFEM2D** command instructs the 2D finite-element EM wave solver to solve the Maxwell equations.

```
EMFEM2D
  [ WaveLength=<<num>> ]   [ Intensity=<<num>> ]
  [ Quan.Eff=<<num>> ] [ SpectrumFile=<<str>> ]
  [ wTE=<<num>> ] [ wTM=<<num>> ] [ phase.TE=<<num>> ] [ phase.
     TM=<<num>> ] [ Angle=<<num>> ]
  [ ABC.Type= ( FirstOrder | SecondOrder | PML ) ]
  [ ABC.Shape= ( Circle | Ellipse | Unknow ) ]
  [ LS = (CGS|BICG|BCGS|GMRES|TFQMR|LU|UMFPACK|SuperLU|
          MUMPS|SuperLU_Dist) ]
  [ PC = (Identity|Jacobian|BJacobian|ASM|ILU) ]
```

Parameters

Parameter	Type	Description	Default	Unit
WaveLength	numeric	The Wave length of the simulation. Synonym: **Lambda**.	0.532	μm
Intensity	numeric	Name of the TIF file to import from.	0.0	W/cm^2
Quan.Eff	numeric	Quantum efficiency for photo-generation.	1.0	none
SpectrumFile	string	Read wavelength, intensity and quantum efficiency from a text file.	none	none
wTE	numeric	The weight of TE mode in the total intensity.	0.5	none
wTM	numeric	The weight of TM mode in the total intensity.	0.5	none
Phase.TE	numeric	The initial phase of the TE mode.	0.0	degree
Phase.TM	numeric	The initial phase of the TM mode.	0.0	degree
Angle	numeric	The direction of incident plane wave, measured in degrees from the +x axis to the wave vector.	90	degree
ABC.Type	enum	The type of absorbing boundary condition used at the outer boundary of the simulation.	**SecondOrder**	none

ABC.Shape	enum	Inform Genius the shape of absorbing boundary. When the shape is known to be **Circle** or **Ellipse**, the calculation of boundary curvature can be accelerated.	**Uknow**	none
LS	enum	Selects the linear solver algorithm.	**BCGS**	none
PC	enum	Selects the linear preconditioner algorithm.	**ILU**	none

Description

The spectrum file is a plain text file with each line for one frequency point in the spectrum. Each line contains 2 or 3 numerical values separated by whitespaces:

```
<wavelength>  <intensity>  [quantum efficiency]
```

Genius will perform FEM EM simulation at each frequency point, calculate the optical generation at each frequency, and sum up the total carrier generation.

When quantum efficiency is given, Genius will use this value to calculate the optical carrier generation rate. Otherwise, Genius will use the following quantum efficiency

$$\eta = \frac{floor(\hbar\nu/E_g) \cdot E_g}{\hbar\nu} \tag{2.10}$$

The sum of weights **wTE** and **wTM** must be 1. When either **wTE** or **wTM** is given, Genius will calculate the other automatically. If both values are given, Genius will check if they add up to unity.

Absorbing Boundary The absorbing boundary condition can be first-order or second-order, and can be selected by the **ABC.Type** parameter. In general, the second-order boundary is preferred. When possible, a circular shape absorbing boundary is highly recommended. The ABC should be at least one wave-length away from the scatterer.

Mesh Size In FEM EM simulation, the mesh size should be $1/16$ to $1/20$ of the optical wavelength in the material. Note that wavelength decreases for materials with relative permittivity greater than 1. As a result, the mesh in dielectric materials (such as Si) should be finer.

RAYTRACE

The **RAYTRACE** command instructs the ray-tracing solver to calculate the light propagation.

```
RAYTRACE
  {
    SpectrumFile=<<str>> |
    ( Lambda=<<num>> Intensity=<<num>> [Quan.Eff=<<num>>] )
  }
  [ Theta=<<num>> ] [ Phi=<<num>> ] [ Ray.Density=<<num>> ]
```

Parameters

Parameter	Type	Description	Default	Unit
SpectrumFile	string	Filename of the spectrum file. This file should describe the wavelength and intensity at each frequency in the spectrum.	**none**	none
Lambda	numerical	Wavelength of the monochromatic light. Synonym: **Wavelength**	**0.532**	μm
Intensity	numerical	Intensity of the monochromatic light.	**0.0**	W/cm^2
Quan.Eff	numerical	Quantum efficiency of carrier generation for the monochromatic light. If not explicitly specified, Genius will automatically calculate the quantum efficiency using the bandgap of the material.	**auto**	none
Theta	numerical	Azimuth angle of the wave vector of the incident light.	0	degree
Phi	numerical	Inclination angle of the wave vector of the incident light, measured from the y-axis.	0	degree
E.Theta	numerical	Azimuth angle of the E-vector of the incident light.	0	degree
E.Phi	numerical	Inclination angle of the E-vector of the incident light, measured from the y-axis.	90	degree
Ray.Density	numerical	Density of light rays. In 2D structures, each mesh element is expected to have at least **Ray.Density** number of rays. In 3D structures, each mesh element has at least **Ray.Density*Ray.Density** number of rays.	10	none

Description

The direction of the incident wave is specified by the azimuth and inclination angle θ and ϕ, and the direction unit vector can be expressed as

$$\vec{r} = \begin{pmatrix} \sin\phi\cos\theta \\ \cos\phi \\ \sin\phi\sin\theta \end{pmatrix} \tag{2.11}$$

The default values of **Theta** and **Phi** corresponds to incidence along the positive y-axis direction.

Ray Seeding Genius seeds the incident light rays automatically. The density of rays is determined by the smallest element in device mesh and the parameter **ray.density**. The ray to ray distance is set to D=$d_{\min}$/**ray.density**. In general, Genius guarantees that each mesh element has at least **ray.density** rays in 2D structures, and **ray.density*ray.density** rays in 3D structures, respectively.

Boundaries When a ray incidents on Ohmic boundaries and Schottky boundaries, the tracing terminates and the incident ray is removed. If a Neumann boundary has the **Reflection** parameter set to true (default is false), the above rule also applies.

On the other hand, the interfaces with metallic (electrode) regions are not affected by the above rule, and refraction/reflection rays are generated.

Absorption When the ray passes though a mesh element, the optical power associated to the ray deposits in this element according to the absorption coefficient. Genius will sum the power deposited by all the rays and calculate the optical carrier generation rate at each node.

Spectrum file With the spectrum file we can specify multi- spectrums of light with different intensity. Spectrum file should contain at lease two spectrums. Each line in the spectrum file should have the following columns

- The wavelength in the unit of micro-meter.
- Intensity density (per unit wavelength) at this wavelength. The unit is W/cm^2/ μm. Note that this differs from the **intensity** parameter for monochromatic light.
- (Optional) quantum efficiency for carrier generation. If not explicitly specified, Genius will automatically calculate the quantum efficiency using the bandgap of the material.

A sample spectrum file is listed below.

```
# lambda      intensity
  0.912       0.2
  0.954       0.5
  1.00        0.4
  1.05        0.3
  1.11        0.2
```

```
1.17        0.2
1.24        0.1
```

Example

```
RayTrace  wavelength=0.4  intensity=0.1  k.phi=0  ray.density=10.0
RayTrace  SpectrumFile=am15a.txt  k.phi=0  ray.density=10.0
```

Input and Output

IMPORT

The **IMPORT** command loads mesh structures and solutions from a file, and builds the simulation system based on it.

```
IMPORT
  { CGNSFile=<<str>>  |  TIFFile=<<str>>  |  TIF3DFile=<<str>>  |
    SilvacoFile=<<str>>  |  ISEFile=<<str>>  }
```

Parameters

Parameter	Type	Description	Default	Unit
CGNSFile	string	Name of the CGNS file to import from.	**none**	none
TIFFile	string	Name of the TIF file to import from.	**none**	none
TIF3DFile	string	Name of the TIF3D file to import from.	**none**	none
SilvacoFile	string	Name of the Silvaco str file to import from.	**none**	none
ISEFile	string	Name of the DF-ISE file to import from.	**none**	none

description

Genius can load device definition and data from many different files. The CFD General Notation System (CGNS) is the native file supported by Genius. User can save the current state of device under simulation by **EXPORT** command and then load it again by **IMPORT** command. Besides the native CGNS format. Genius can also load device structure from Technology Interchange Format (TIF) format which is commonly used by Synopsys medici and tsuprem, DF-ISE format used by Synopsys Sentaurus, structure format used by Silvaco, and a externed TIF3D file generated by Gds2mesh tool.

Example

```
IMPORT     CGNSFile = "foo.cgns"
IMPORT     TIFFile  = "foo.tif"
```

EXPORT

The **EXPORT** command saves mesh structures and solutions to a file.

```
EXPORT
  [ CGNSFile=<<str>> ] [ VTKFile=<<str>> ] [ ISEFile=<<str>> ]
  [ BCInfo=<<str >>]
  { [ NodeInfo=<<str >>] [ LUnit = (m|cm|um|nm) ]
    [ Numbering=<<bool>> ] }
```

Parameters

Parameter	Type	Description	Default	Unit
CGNSFile	string	Export the mesh and data in the CGNS file with the supplied file name.	**none**	none
VTKFile	string	Export the mesh and data in the VTK file with the supplied file name.	**none**	none
ISEFILE	string	Export device mesh and doping information in the DF-ISE format into two files with supplied file name and .grd and .dat file extersion, respectively.	**none**	none
BCFile	string	Export the boundary condition definition of the structure to the file, which can be loaded in another Genius input file with an **INCLUDE** command.	**none**	none
NodeFile	string	Export the coordinates of the mesh points to a plain text file with the supplied file name.	**none**	none
LUnit	enum	Length unit used for node coordinates, when exporting to **NodeFile**.	**um**	none
Numbering	bool	Whether to print node number in **NodeFile**.	**true**	none

description

Genius can save device definition and data to files in the CFD General Notation System (CGNS) which is natively supported. Genius also saves mesh and solution data with the Visualization ToolKit (VTK) formats which is supported by VisualTCAD and many other post-process tool such as Paraview and Visit. Since version 1.7, Genius can export device mesh and doping information in the DF-ISE format.

Example

```
EXPORT    CGNSFile = "foo.cgns"
EXPORT    CGNSFile = "foo.cgns" VTKFile="bar.vtk"
EXPORT    ISEFile = "foo"
```

HOOK

The **HOOK** command loads or unloads hook functions, which are called before and after solving the device equations. Various hook functions are available that serve as pre-/post-processors, input/output formatters and interface with other softwares.

```
HOOK
  [ Load=<<str>> [ real<<<str>>>=<<num>> [...] ] [ str<<<str
      >>>=<<str>> [...] ]
  [ Unload=<<str>> ]
```

Parameters

Parameter	**Type**	**Description**	**Default**	**Unit**
Load	string	Name of the hook function to be loaded.	**none**	none
real $< $ **str** $>$	numerical	Custom numerical parameters for the loaded hook function.	**none**	none
string $<$ **str** $>$	string	Custom string parameters for the loaded hook function.	**none**	none
Unload	enum	Name of the hook function to be unloaded.	**none**	none

Description

The **HOOK** command should be invoked before the **SOLVE** command to be effective. Some hooks are only compatible with some of the analysis modes (e.g. with DC Sweep mode only). After a hook is loaded, it is effective in subsequent **SOLVE** commands as well if the hook function is compatible with the analysis mode, until it is unloaded.

Some hooks accepts custom parameters when the user loads them. If one tries to load a previously loaded hook function again, the parameters specified during previous loading are all erased.

Default Hook Functions

The following hook functions are loaded by default.

- **gnuplot**. Save the terminal voltages and currents from the simulation result in plain text format, which is compatible with the *gnuplot* ploting program. This hook is
- **raw**. Save the terminal voltages and currents from the simulation result in the SPICE raw format. The output file has the extention *.raw*.

Additionally, a few optional hook functions are available, and the user can load some of them with the **HOOK** command. In the following sections, we briefly describe each of them.

CV Postprocessor

The **cv** hook function extracts the capacitance of gate electrodes. One can load this postprocessor with the command

```
HOOK    Load=cv
```

and the extracted capacitance-voltage data will be saved in output file with the extension *.cv*. The electric charge in the gate electrodes are first integrated, and capacitance is then extracted by differentiating with respect to the scanning voltage. It works in the DC voltage sweep mode only and the extracted capacitance is quasi-static. For capacitance analysis at high frequency, one should use the AC analysis mode.

Probe Postprocessor

The **probe** hook function allows the user to monitor the potential and carrier concentration at a particular point in the device. It can be loaded with the following command

```
HOOK    Load=probe  real<x>=1.0  real<y>=1.2
```

where the following custom parameters are recognized.

Parameter	Type	Description	Default	Unit
x	numerical	The x-coordinate of the probe.	0	μm
y	numerical	The y-coordinate of the probe.	0	μm
z	numerical	The z-coordinate of the probe.	0	μm

Visualization and Animation Postprocessor

The vtk hook function allows the user to save VTK file at each step in DC and transient simulation. The serial of VTK files which contains field evolving information can be viewed by VisualTCAD, Paraview or Visit. Furthermore, user can make animation with the serial of VTK files with Paraview or Visit.

Interface to Visit visualizer

Genius can feed solution to the visualizer program Visit through the **visit** hook function.

CHAPTER 3

Extending Genius Simulator

The Genius device simulator can be extended by users in several ways.

- Through the physical model interface (PMI), new materials can be added and existing materials can be modified.
- Pre-processing and post-processing steps can be added through the hook interface.
- The simulator is programmable through its Python interface.

This chapter outlines the procedure of extending Genius with the various interfaces. This chapter is intended for advanced readers who intend to write programs to customize or extend the Genius simulator. An intermediate level of knowledge about the C++ programming language and the development tools in Linux or Windows.

Custom Material Models

Overview of PMI

Each material in Genius is packaged as a dynamically linked library (DLL). With this design, new materials can be added, and existing materials modified, without the need of re-building the entire simulator.

Files

The material library files are located in the directory *$GENIUS_DIR/lib*. The filename for material foo is *libfoo.so*.

Material Catalog File

In addition to the material's DLL, an entry in the catalog file *$GENIUS_DIR/lib/material.def* is needed for the material model to function properly in Genius. Below is a few entries in *material.def*.

```
Si
{
    property = Semiconductor
    alias    = Silicon
    color    = 0xffb6c1ff
}

GaAs
{
    property = Semiconductor
    color    = 0xeaeaeaff
}

AlGaAs
{
    # mole dependence: Ga(1-x)Al(x)As
    property = SingleCompoundSemiconductor
    color    = 0xa6d6d6ff
}

SiO2
{
    property = Insulator
    alias    = Ox
    alias    = Oxide
    color    = 0x7d0505ff
}
```

Each entry corresponds to a material. An entry starts with material's ID and the content of the entry is contained in the pair of curly braces. The content is a list of parameter=value pairs, each in its own line. The parameter **property** specifies the type of the material, the possible values are

- **Semiconductor**: Elemental and simple compound semiconductors with fixed compositions.
- **SingleCompoundSemiconductor**: Compound semiconductors with a single mole fraction number.
- **ComplexCompoundSemiconductor**: Complex compound semiconductors, with more than one mole fraction number.
- **Conductor**: metals and metallic poly-silicon
- **Vacuum**: only used in optical simulation
- **PML**: only used in optical simulations

Each material can have several aliases in addition to its ID. In the **Material** parameter of the **REGION** command, one can use either the aliases or the ID of a material.

Classes

Figure 3.1: Class hierarchy of the PMI

The class hierarchy of the PMI interface is shown in figure **??**, p. 116. The abstract base class PMI provides the basic facilities for all physical models. For each category of materials, semiconductor, conductor or insulator, there is a class derived from PMI (in the second column). For a semiconductor material, there are several 7 aspects of its physics. Every aspect is described in a class, as shown in the third column. For insulators and conductors, the physics are simpler. The physics covered by each class is summarized below.

- BasicParameter: mass density, dielectric permittivity, magnetic permeability and electron affinity.
- BandStructure: bandgap, carrier effective mass, effective density of states, generation/recombination, thermoionic emission and (temporarily) band-to-band tunneling.
- Mobility: carrier mobility, include its degradation at surfaces and carrier velocity saturation.
- Avalanche: impact ionization generation.
- Trap: carrier capture and emission at trap states.
- Thermal: thermal conductivity and thermal capacity.
- Optical: refraction index (real and imaginary parts) at different wavelengths.

All the above classes are abstract and can not be instantiated. When one implements a material, one must derive from each of the above classes, and provide a concrete implementation. For example, for the 4H-SiC material, the classes in the last column of figure **??**, p. 116 are implemented and packed in the dynamic library *libSiC4H.so*.

Since dynamic libraries has a C interface (instead of a C++ one), a creator function with C-style name and calling convention, must accomany each concrete class in the dynamic library. For example, the C function PMIS_SiC4H_BandStructure_Defa in the library *libSiC4H.so* is responsible for creating an instance of the class GSS_SiC4H_BandStructure.

How a PMI Library is Loaded

We shall describe the loading sequence of physical models, assuming the following two commands appear in the input file to Genius.

```
# MESH commands
# ...
REGION Label=Substrate Material=Silicon
REGION Label=GOX Material=SiO2
#------------------
```

```
# ...
PMI Region=Substrate Type=Mobility Model=HP
PMI Region=Substrate Type=Avalanche Model=HP
```

The **REGION** commands are read in first, together with the meshing commands. After all the structure specification commands are read, the initial simulation system is built. For the region **Substrate**, the default material models for silicon are loaded. For the region GOX, the default models for SiO2 are loaded.

After the simulation system is built, the rest of the commands, including those **PMI** commands are executed sequentially. When the first PMI command is executed, the region **Substrate** is looked up and found to be of material silicon. The dynamic library *libSi.so* is loaded. From the material name, the model type Mobility, the model name HP, we assemble the creator function's name PMIS_Si_Mob_HP. This creator function is called, which returns an object instance of the class GSS_Si_Mob_HP. This GSS_Si_Mob_HP class, as a derived class of PMIS_Mobility, contains routines for calculating carrier mobility and velocity saturation.

How a PMI Library is Used

When the Genius simulator assemble the Poisson and continuity equations set out in Level 1 Drift-Diffusion Equation, p. 10, material properties, such as doping concentration and mole fraction are available only at each of the grid nodes. Many derived physical quantities, such as carrier mobility and SRH recombination rate, are evaluated at each mesh node as well. This is best illustrated with an example.

Suppose when we assemble the carrier continuity equation, we need to calculate the bandgap narrowing due to heavy doping at a grid node. We know the material of the current mesh node is 4H-SiC, and from figure **??**, p. 116, the PMI class that handles the bandgap narrowing physics is GSS_SiC4H_BandStructure. In fact, as a sub-class of PMIS_BandStructure, it has a method EgNarrow that calculates bandgap narrowing. The definition of the method is shown below, also shown is the method nie that calculates the effective intrinsic carrier concentration.

```
class GSS_SiC4H_BandStructure : public PMIS_BandStructure
{
  // ...

public:
  double EgNarrow(const double &p, const double &n,
                  const double &Tl)
  {
    double Na = ReadDopingNa();
    double Nd = ReadDopingNd();
    double N = Na+Nd+1.0*std::pow(cm,-3);
    double x = log(N/N0_BGN);
    return V0_BGN*(x+sqrt(x*x+CON_BGN));
  }

  double nie (const double &p, const double &n,
              const double &T1)
  {
    double bandgap = Eg(T1);
    double Nc = NC300*std::pow(T1/T300,NC_F);
    double Nv = NV300*std::pow(T1/T300,NV_F);
    return sqrt(Nc*Nv)*exp(-bandgap/(2*kb*T1))
             *exp(EgNarrow(p, n, T1));
  }

  // ...
};
```

It is obvious that this few lines implements the Slotboom model of bandgap narrowing

$$\Delta E_g = \frac{E_{\text{bgn}}}{2k_bT}\left[\ln\frac{N_{\text{total}}}{N_{\text{ref}}} + \sqrt{\left(\ln\frac{N_{\text{total}}}{N_{\text{ref}}}\right)^2 + C_{\text{bgn}}}\right]. \tag{3.1}$$

In this simple model, neither the electron, hole concentration nor the lattice temperature affects the bandgap narrowing. Therefore the arguments are ignored. The model chiefly relies on the doping concentration. The ReadDopingNa and ReadDopingNd methods, inherited from the PMIS_Server class, returns the acceptor and donor concentrations at the present mesh node, respectively.

It is also necessary to evaluate the partial derivatives of the bandgap narrowing, with respective to all unknowns in the system of equations. This may sound an involved task. However, Genius uses the automatic differentiation technology which hides most of the complications from the user. The following snippet shows the method that calculates the partial derivatives. It is also named EgNarrow, but note the difference in the type of arguments.

```
class GSS_SiC4H_BandStructure : public PMIS_BandStructure
{
  // ...

public:

  AutoDScalar EgNarrow(const AutoDScalar &p, const AutoDScalar &
     n,
                       const AutoDScalar &T1)
  {
    double Na = ReadDopingNa();
    double Nd = ReadDopingNd();
    double N = Na+Nd+1.0*std::pow(cm,-3);
    double x = log(N/N0_BGN);
    return V0_BGN*(x+sqrt(x*x+CON_BGN));
  }
  AutoDScalar nie (const AutoDScalar &p, const AutoDScalar &n,
                   const AutoDScalar &Tl)
  {
    AutoDScalar bandgap = Eg(Tl);
    AutoDScalar Nc = NC300*adtl::pow(Tl/T300,NC_F);
    AutoDScalar Nv = NV300*adtl::pow(Tl/T300,NV_F);
    AutoDScalar ret=sqrt(Nc*Nv)*exp(-bandgap/(2*kb*Tl))
                       *exp(EgNarrow(p, n, Tl));
    return ret;
  }
  // ...

};
```

It is clear that the "derivative" version of EgNarrow and nie are almost identical to the "normal" version. In almost all situations, it is sufficient to replace the scalar type double with the class type AutoDScalar.

Procedure of Building a Custom Material Library

The procedure of building a custom material library is outlined in this section.

Building the Example

Step 1 Run the configure script.

```
cd examples/Material
./configure.sh
```

Step 2 Enter the material directory and modify the source files as needed.

```
cd 4H-SiC
```

Step 3 Build the material library.

```
make
```

Step 4 Copy the library file *libSiC4H.so* to *$GENIUS_DIR/lib/*, backing up the old file is strongly recommended. The new material library is ready for use now.

Create a New Material Library

Step 1 Make a copy of some existing model.

```
cp -R 4H-SiC 6H-SiC
```

Step 2 Replace the material name string in the all source files, with the new material name, and rename the files.

```
sed −i −e "s/SiC4H/SiC6H/g" makefile
sed −i −e "s/SiC4H/SiC6H/g" ∗.cc
rename "s/4H−SiC/6H−SiC/" ∗.cc
```

Step 3 Edit the source, and build the material library.

```
cd 6H–SiC
make
```

Step 4 Copy the library file *libSiC4H.so* to *$GENIUS_DIR/lib/*, and edit the catalog file $GENIUS_DIR/lib/material.def

```
SiC6H
{
    property = Semiconductor
    alias    = 6HSiC
    color    = 0xffb6c1ff
}
```

The new material library is ready for use now.

Class References

The class reference presented here is a selected and adapted version of the classes declared in the PMI development kit. Some macros are expanded and some internal details are omitted, so that this guide is easier to follow.

PMI_Server Class

```
PMI_Server(const PMI_Environment &env);
```

Constructor.

```
void ReadCoordinate (double& x, double& y, double& z) const
```

Return the spatial coordinates of the mesh node.

```
double ReadTime() const
```

Return the current time in simulation. This is only meaningful in transient-mode simulations.

```
double ReadUserScalarValue (std::string &name) const
```

Return the scalar value with the given name at the current node. An examples of the scalar values at a node is the density of trap state.

PMIS_Server Class

```
double ReadxMoleFraction () const
```

Return the mole fraction x at the current node. Only used in compound semiconductor with variable compositions.

```
double ReadxMoleFraction (const double mole_xmin,
                          const double mole_xmax) const
```

Return the mole fraction x at the current node, with upper and lower limits.

```
double ReadyMoleFraction () const
```

Return the mole fraction y at the current node. Only used in compound semiconductor with variable compositions.

```
double ReadyMoleFraction (const double mole_ymin,
                          const double mole_ymax) const
```

Return the mole fraction y at the current node, with upper and lower limits.

```
double ReadDopingNa () const
```

Return the acceptor dopant concentration.

```
double ReadDopingNd () const
```

Return the donor dopant concentration.

PMIS_BasicParameter Class

```
virtual double Density (const PetscScalar &Tl) const = 0
```

Return the mass density g/cm^{-3} of the material.

```
virtual double Permittivity () const = 0
```

Return the relative permittivity of the material.

```
virtual double Permeability () const = 0
```

Return the relative permeability of the material.

```
virtual double Affinity (const double &Tl) const = 0
```

Return the affinity energy [eV] of the material.

PMIS_BandStructure Class

Unless otherwise stated, the arguments of the following methods are

- p hole concentration
- n electron concentration
- Tl lattice temperature
- Tp hole temperature
- Tn electron temperature

```
virtual double      Eg (const PetscScalar &Tl) = 0
virtual AutoDScalar Eg (const AutoDScalar &Tl) = 0
```

Return the band gap of semiconductor.

```
virtual double      EgNarrow (const double      &p,
                              const double      &n,
                              const double      &Tl) = 0
virtual AutoDScalar EgNarrow (const AutoDScalar &p,
                              const AutoDScalar &n,
                              const AutoDScalar &Tl) = 0
```

Return the band gap narrowing due to heavy doping.

```
virtual double      EgNarrowToEc (const double      &p,
                                  const double      &n,
                                  const double      &Tl) = 0
virtual AutoDScalar EgNarrowToEc (const AutoDScalar &p,
                                  const AutoDScalar &n,
                                  const AutoDScalar &Tl) = 0
```

Return the conduction band shift due to band gap narrowing.

```
virtual double      EgNarrowToEv (const double      &p,
                                  const double      &n,
                                  const double      &Tl) = 0
virtual AutoDScalar EgNarrowToEv (const AutoDScalar &p,
                                  const AutoDScalar &n,
                                  const AutoDScalar &Tl) = 0
```

Return the valence band shift due to band gap narrowing.

```
virtual double      EffecElecMass (const double      &Tl) = 0
virtual AutoDScalar EffecElecMass (const AutoDScalar &Tl) = 0
```

Return the effective mass of electrons (DOS mass).

```
virtual double      EffecHoleMass (const double      &Tl) = 0
virtual AutoDScalar EffecHoleMass (const AutoDScalar &Tl) = 0
```

Return the effective mass of holes (DOS mass).

```
virtual double      Nc (const double     &Tl) = 0
virtual AutoDScalar Nc (const AutoDScalar &Tl) = 0
```

Return the effective density of states in the conduction band.

```
virtual double      Nv (const double     &Tl) = 0
virtual AutoDScalar Nv (const AutoDScalar &Tl) = 0
```

Return the effective density of states in the valence band.

```
virtual double ni ( const PetscScalar &Tl) = 0
```

Return the intrinsic carrier concentration.

```
virtual double      nie (const double      &p,
                         const double      &n,
                         const double      &Tl) = 0
virtual AutoDScalar nie (const AutoDScalar &p,
                         const AutoDScalar &n,
                         const AutoDScalar &Tl) = 0
```

Return the effective intrinsic carrier concentration.

```
virtual PetscScalar TAUN (const double &Tl) = 0
virtual PetscScalar TAUP (const double &Tl) = 0
```

Return the minority carrier life time due to SHR recombination.

```
virtual double      R_Direct (const double      &p,
                              const double      &n,
                              const double      &Tl) = 0
virtual AutoDScalar R_Direct (const AutoDScalar &p,
                              const AutoDScalar &n,
                              const AutoDScalar &Tl) = 0
```

Return the direct (interband) recombination rate.

```
virtual double      R_Auger (const double      &p,
                             const double      &n,
                             const double      &Tl) = 0
virtual AutoDScalar R_Auger (const AutoDScalar &p,
                             const AutoDScalar &n,
                             const AutoDScalar &Tl) = 0
```

Return the Auger recombination rate.

```
virtual double      R_Auger_N (const double      &p,
                               const double      &n,
                               const double      &Tl) = 0
virtual AutoDScalar R_Auger_N (const AutoDScalar &p,
                               const AutoDScalar &n,
                               const AutoDScalar &Tl) = 0
```

```
virtual double      R_Auger_P (const double      &p,
                               const double      &n,
                               const double      &Tl) = 0
virtual AutoDScalar R_Auger_P (const AutoDScalar &p,
                               const AutoDScalar &n,
                               const AutoDScalar &Tl) = 0
```

Return the rates of electron and hole initiated Auger recombination.

```
virtual double      R_SHR (const double      &p,
                           const double      &n,
                           const double      &Tl) = 0
virtual AutoDScalar R_SHR (const AutoDScalar &p,
                           const AutoDScalar &n,
                           const AutoDScalar &Tl) = 0
```

Return the rate of SRH recombination.

```
virtual PetscScalar R_Surf (const double      &p,
                            const double      &n,
                            const double      &Tl,
                            const double      &len) =0;
virtual AutoDScalar R_Surf (const AutoDScalar &p,
                            const AutoDScalar &n,
                            const AutoDScalar &Tl,
                            const PetscScalar &len) =0;
```

Return the rate of SRH recombination.

```
virtual double      R_SHR (const double      &p,
                           const double      &n,
                           const double      &Tl) = 0
virtual AutoDScalar R_SHR (const AutoDScalar &p,
                           const AutoDScalar &n,
                           const AutoDScalar &Tl) = 0
```

Return the rate of SRH and surface recombination at semiconductor-insulator interface.

```
virtual double      Recomb (const double      &p,
                            const double      &n,
                            const double      &Tl) = 0
virtual AutoDScalar Recomb (const AutoDScalar &p,
                            const AutoDScalar &n,
                            const AutoDScalar &Tl) = 0
```

Return the total recombination rate of all recombination mechanisms.

```
virtual double      ElecEnergyRelaxTime(const double      &Tn,
                                        const double      &Tl) =
                                            0
virtual AutoDScalar ElecEnergyRelaxTime(const AutoDScalar &Tn,
                                        const AutoDScalar &Tl) =
                                            0
```

Return the electron energy relaxation time. Used in EBM simulations.

```
virtual double      HoleEnergyRelaxTime(const double      &Tn,
                                        const double      &Tl) =
                                          0
virtual AutoDScalar HoleEnergyRelaxTime(const AutoDScalar &Tn,
                                        const AutoDScalar &Tl) =
                                          0
```

Return the hole energy relaxation time. Used in EBM simulations.

```
virtual double      SchottyJsn (double      n,
                                double      Tl,
                                double      Vb) = 0
virtual AutoDScalar SchottyJsn (AutoDScalar n,
                                AutoDScalar Tl,
                                AutoDScalar Vb) = 0
```

Return the electron current density at Schottky contacts. The argument **Vb** is the electron barrier height.

```
virtual double      SchottyJsp (double      n,
                                double      Tl,
                                double      Vb) = 0
virtual AutoDScalar SchottyJsp (AutoDScalar n,
                                AutoDScalar Tl,
                                AutoDScalar Vb) = 0
```

Return the hole current density at Schottky contacts. The argument **Vb** is the electron barrier height.

```
virtual double SchottyBarrierLowerring (double      eps,
                                        double      E) = 0
```

Return the total recombination rate of all recombination mechanisms.

```
virtual double ARichN() = 0
virtual double ARichP() = 0
```

Return the Richardson constant for electrons and holes, respectively.

```
virtual double      ThermalVn (double      Tl) = 0
virtual AutoDScalar ThermalVn (AutoDScalar Tl) = 0
```

Return the electron thermal emission velocity at hetero-junctions

```
virtual double      ThermalVp (double      Tl) = 0
virtual AutoDScalar ThermalVp (AutoDScalar Tl) = 0
```

Return the Richardson constant for electrons and holes, respectively.

```
virtual double      BB_Tunneling(const double      &Tl,
                                 const double      &E) = 0
virtual AutoDScalar BB_Tunneling(const AutoDScalar &Tl,
                                 const AutoDScalar &E) = 0
```

Return the band-to-band tunneling generation rate. The argument $\mathbf{E}$ is the magnitude of electric field.

PMIS_Mobility

Unless otherwise stated, the arguments of the following methods are

- $\mathbf{p}$ hole concentration
- $\mathbf{n}$ electron concentration
- $\mathbf{Tl}$ lattice temperature
- $\mathbf{Tp}$ hole temperature
- $\mathbf{Tn}$ electron temperature

```
virtual double      ElecMob (const double      &p,
                             const double      &n,
                             const double      &Tl,
                             const double      &Ep,
                             const double      &Et,
                             const double      &Tn) const=0
virtual AutoDScalar ElecMob (const AutoDScalar &p,
                             const AutoDScalar &n,
                             const AutoDScalar &Tl,
                             const AutoDScalar &Ep,
                             const AutoDScalar &Et,
                             const AutoDScalar &Tn) const=0
```

Return the electron mobility. The argument $\mathbf{Ep}$ and $\mathbf{Et}$ are the parallel and transverse components of the electric field, respectively.

```
virtual double      HoleMob (const double      &p,
                             const double      &n,
                             const double      &Tl,
                             const double      &Ep,
                             const double      &Et,
                             const double      &Tp) const=0
virtual AutoDScalar HoleMob (const AutoDScalar &p,
                             const AutoDScalar &n,
                             const AutoDScalar &Tl,
                             const AutoDScalar &Ep,
                             const AutoDScalar &Et,
                             const AutoDScalar &Tp) const=0
```

Return the hole mobility. The argument $\mathbf{Ep}$ and $\mathbf{Et}$ are the parallel and transverse components of the electric field, respectively.

PMIS_Avalanche

Unless otherwise stated, the arguments of the following methods are

- **Tl** lattice temperature
- **Tp** hole temperature
- **Tn** electron temperature
- **Ep** driving force of impact ionization (parallel electric field).
- **Eg** bandgap

```
virtual double      ElecGenRate (const double     &Tl,
                                 const double     &Ep,
                                 const double     &Eg) const =
                                     0
virtual AutoDScalar ElecGenRate (const AutoDScalar &Tl,
                                 const AutoDScalar &Ep,
                                 const AutoDScalar &Eg) const =
                                     0
```

Return the generation rate of electron-initiated impact ionization.

```
virtual double      HoleGenRate (const double     &Tl,
                                 const double     &Ep,
                                 const double     &Eg) const =
                                     0
virtual AutoDScalar HoleGenRate (const AutoDScalar &Tl,
                                 const AutoDScalar &Ep,
                                 const AutoDScalar &Eg) const =
                                     0
```

Return the generation rate of hole-initiated impact ionization.

```
virtual double      ElecGenRateEBM (const double     &Tn,
                                    const double     &Tl,
                                    const double     &Eg) const
                                        = 0
virtual AutoDScalar ElecGenRateEBM (const AutoDScalar &Tn,
                                    const AutoDScalar &Tl,
                                    const AutoDScalar &Eg) const
                                        = 0
```

Return the generation rate of electron-initiated impact ionization, calculate from the electron temperature. Used in EBM simulation only.

```
virtual double      HoleGenRateEBM (const double     &Tp,
                                    const double     &Tl,
                                    const double     &Eg) const
                                        = 0
virtual AutoDScalar HoleGenRateEBM (const AutoDScalar &Tp,
                                    const AutoDScalar &Tl,
                                    const AutoDScalar &Eg) const
                                        = 0
```

Return the generation rate of hole-initiated impact ionization, calculate from the electron temperature. Used in EBM simulation only.

PMIS_Trap

[[TODO]]

PMIS_Thermal

The argument Tl is the lattice temperature.

```
virtual double      HeatCapacity  (const double      &Tl) const
   = 0
virtual AutoDScalar HeatCapacity  (const AutoDScalar &Tl) const
   = 0
```

Return the heat capacity $JK^{-1}cm^{-3}$ of the material.

```
virtual double      HeatConduction(const double      &Tl) const
   = 0
virtual AutoDScalar HeatConduction(const AutoDScalar &Tl) const
   = 0
```

Return the heat conduction $W/cm/K$ of the material.

PMIS_Optical

The argument lambda is the wavelength, Eg the bandgap, and Tl the lattice temperature.

```
virtual std::complex<double> RefractionIndex(double lamda,
                                             double Eg,
                                             double Tl) const =
                                                0
```

Return the heat capacity J Kcm^{-3} of the material.

PMII

empty

PMII_BasicParameter

See PMIS_BasicParameter Class, p. 121.

PMII_Thermal

See PMIS_Thermal, p. 128.

PMII_Optical

See PMIS_Optical, p. 128.

PMIC

empty

PMIC_BasicParameter

```
virtual double Density (const PetscScalar &Tl) const = 0
```

Return the mass density g/cm^{-3} of the material.

```
virtual double Permittivity () const = 0
```

Return the relative permittivity of the material.

```
virtual double Permeability () const = 0
```

Return the relative permeability of the material.

```
virtual double Affinity (const double &Tl) const = 0
```

Return the affinity energy [eV] of the material.

```
virtual double Conductance () const = 0
```

Return the electrical conductance of the material.

PMIC_Thermal

See PMIS_Thermal, p. 128.

PMIC_Optical

See PMIS_Optical, p. 128.

Hooks

[[TODO]]

Python Interface

[[TODO]]

HTTP Frontend

[[TODO]]