ISE TCAD Release 10.0

DESSIS™

Integrated Systems Engineering
Zurich, Switzerland

ISE Integrated Systems Engineering AG
Affolternstrasse 52
CH-8050 Zurich
Switzerland
Phone: +41 44 389 9700
Fax: +41 44 389 9797
www.ise.ch | www.ise.com

Preface

About this manual

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

The manual is divided into parts:

- Part I is an overview of DESSIS. It contains information about how to start DESSIS, the main features of DESSIS, its input file, and mixed-mode DESSIS.
- Part II describes the physics in DESSIS.
- Part III describes the physical models used in laser and LED simulations.
- Part IV presents the automatic grid generation facility and provides background information on the numeric methods used in DESSIS.
- Part V describes the physical model interface (PMI), which provides direct access to certain models in the semiconductor transport equations and the numeric methods in DESSIS.

Typographic conventions

Convention	**Explanation**
< >	Angle brackets
{ }	Braces
[ ]	Brackets
( )	Parentheses
Blue text	Identifies a cross-reference (only on the screen).
Bold text	Identifies a selectable icon, button, menu, or tab. It also indicates the name of a field, window, dialog box, or panel.
`Courier font`	Identifies text that is displayed on the screen or that the user must type. It identifies the names of files, directories, paths, parameters, keywords, and variables.
Italicized text	Used for emphasis, the titles of books and journals, and non-English words. It also identifies components of an equation or a formula, a placeholder, or an identifier.
Menu > **Command**	Indicates a menu command, for example, **File** > **New** (from the **File** menu, select **New**).
NOTE	Identifies important information.

Comments about this manual

To improve ISE technical documentation and allow users to fully explore ISE TCAD™ software, ISE requests the opinions of users about the contents of this manual. ISE welcomes comments and notification of any errors to `manuals@ise.ch`.

ISE Technical Support

ISE Technical Support provides timely and efficient responses to customer requests:

Europe (and the rest of the world):	`support.eu@ise.com`
North America and South America:	`support.us@ise.com`
Japan:	`support.jp@ise.com`
Taiwan:	`support.tw@ise.com`
Korea:	`support.kr@ise.com`
China:	`support.cn@ise.com`

All data submitted to ISE Technical Support is treated confidentially. Upon request, PGP®/GPG encryption of transferred data can be used.

To avoid difficulties with email transmissions and shorten the response time, process files before submitting them. It is preferred that gzip'ed archive files are sent (the procedure is described below). Do not attach large files to an email message. Instead, it is recommended that you place them in the incoming directory on the ISE anonymous FTP server (the procedure is described below). Anonymous users can write to this directory, but cannot read it.

The following information is required:

- Company name or customer number.
- A clear, precise description of the problem.
- Input files to reproduce the problem if required.
- ISE TCAD tool name and version number. Use the option `-v`:

  ```
  $ <tool> -v
  ```

  For example, for a problem with the DESSIS™, use: `$ dessis -v`
- Other information regarding the platform used. This is obtained by using the diagnostic option:

  ```
  $ <tool> -@diag
  ```

  This generates a diagnostics list with extensive system information, including the location and availability of shared libraries.
- For problems that may relate to installation or license issues, provide the output of the command:

  ```
  $ <tool> -@ldiag
  ```

  For example, `$ dessis -@ldiag`. This generates a diagnostics list that includes license information.

- For problems related to Framework tools, start the tool with the option `-verbose`, for example:

  ```
  $ GENESISe -verbose
  ```

- For problems that are difficult to reproduce or take a long time to run, include log files and output files.

Windows® zipped file

Save all relevant files to a directory. Create a zip file of the directory by using a zip tool such as WinZip®.

Gzip'ed tar'ed file

Save all relevant files to a directory. Create a gzip'ed archive of the directory.

When submitting a GENESISe™ project, tar the entire directory to ensure that all files (including those with file names starting with a dot) are included, for example:

```
tar cvf support.tar directory_name
gzip support.tar
```

Procedure for FTP

Files placed on the FTP server must be named `companyname_problem.tar.gz` or `companyname_problem.zip` (for example, `firetexinc_problem.tar.gz`). Refer to this name in the email message sent to ISE Technical Support:

Server:	`ftp.ise.ch` or `ftp.ise.com`
Login:	`anonymous`
Password:	email address
Directory:	`incoming`

If you do not have a GUI-based FTP program, use the built-in FTP program:

1. Open a shell (for Windows®, use command prompt).
2. Type: `ftp ftp.ise.ch`.
3. Login: `anonymous`
4. Change to the incoming directory by typing: `cd incoming`.
5. Change the transfer mode to binary by typing: `bin`.
6. Place the zipped file in the directory by typing: `put companyname_problem.tar.gz`.
7. Exit the server by typing: `quit`.

For assistance using this software and for further information about sending requests to ISE Technical Support, visit `www.ise.ch | www.ise.com`.

Part I Overview

This part of the DESSIS manual contains the following chapters:

CHAPTER 1 Getting started

1.1 About DESSIS

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

NOTE The terms 'grid' and 'mesh' are interchangeable.

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

- The grid (or geometry) file contains a description of the various regions of the device, that is, boundaries, material types, and the locations of any electrical contacts. This file also contains the grid (the locations of all the discrete nodes and their connectivity). The typical contents of this file, such as the boundary and grid of a typical MOSFET structure, are shown in Figure 15.1.

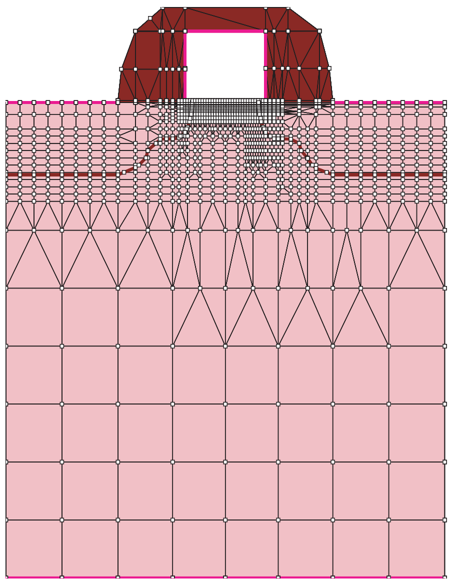


Figure 15.1 Region boundaries of a MOSFET structure with nodes and mesh

- The data (or doping) file contains the properties of the device, such as the doping profiles, in the form of data associated with the discrete nodes. Figure 15.2 is an example. By default, a device simulated in 2D is assumed to have a 'thickness' in the third dimension of 1 µm.

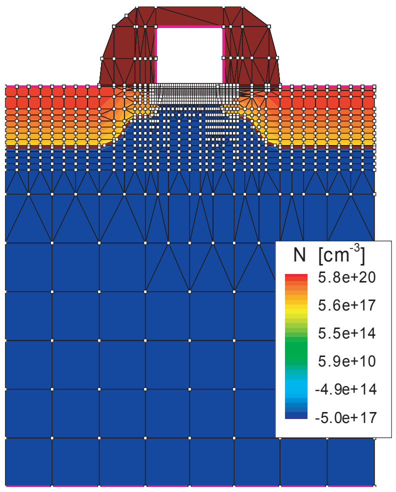


Figure 15.2 Two-dimensional doping profile that is discretized on the nodes of simulation grid

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

- An extensive set of models for device physics and effects in semiconductor devices (drift-diffusion, thermodynamic, and hydrodynamic models).
- General support for different device geometries (1D, 2D, 3D, and 2D cylindrical).
- A extensive set of nonlinear solvers.
- A mixed-mode support of electrothermal netlists with mesh-based device models and SPICE circuit models.

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

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

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

The mixed device and circuit capabilities give DESSIS the ability to solve three basic types of problem: single device simulations, single device with a circuit netlist simulations, and multiple devices with a circuit netlist simulations (see Figure 15.3 on page 15.5).

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

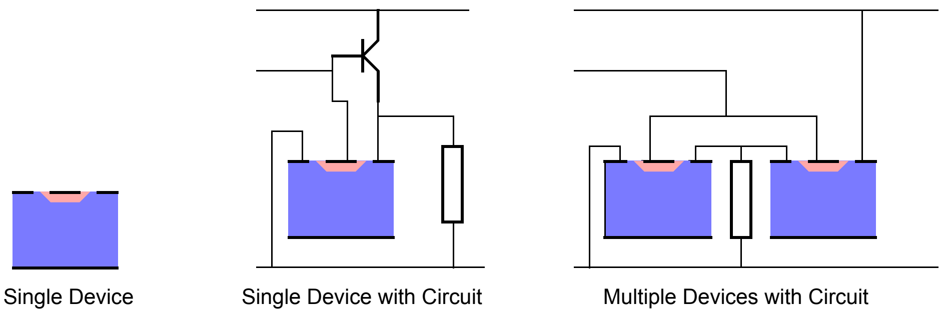


Figure 15.3 Three types of simulation

1.1.1 Creating and meshing device structures

Device structures can be created in various ways, including 1D, 2D, or 3D process simulation (DIOS™), 3D process emulation (DEVISE™), and 2D (MDRAW™ and DEVISE) or 3D (DIP™ and DEVISE) structure editors.

Regardless of the means used to generate a virtual device structure, it is recommended that the structure be remeshed using MDRAW (2D meshing with an interactive graphical user interface (GUI)) or MESH™ (1D, 2D, and 3D meshing without a GUI) to optimize the grid for efficiency and robustness. All device structures used as examples in this section were created and meshed using MDRAW.

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

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

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

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

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

1.1.2 Design flow

A typical device 'design flow' (or 'tool flow' in GENESISe™) involves the creation of a device structure by a process simulation (DIOS) followed by remeshing using MDRAW (for 2D studies). In this scheme, control of mesh refinement is handled automatically through the file `_mdr.cmd` (created by DIOS).

For the following examples, no process simulation is required because MDRAW is used to build the device structures (using analytic doping profiles) and create a suitable mesh.

DESSIS is used to simulate the electrical characteristics of the device. Such a seamless flow through ISE TCAD tools, with the associated file types, is represented in Figure 15.4. Finally, Tecplot-ISE is used to visualize the output from the simulation in 2D, and INSPECT™ is used to plot the electrical characteristics.

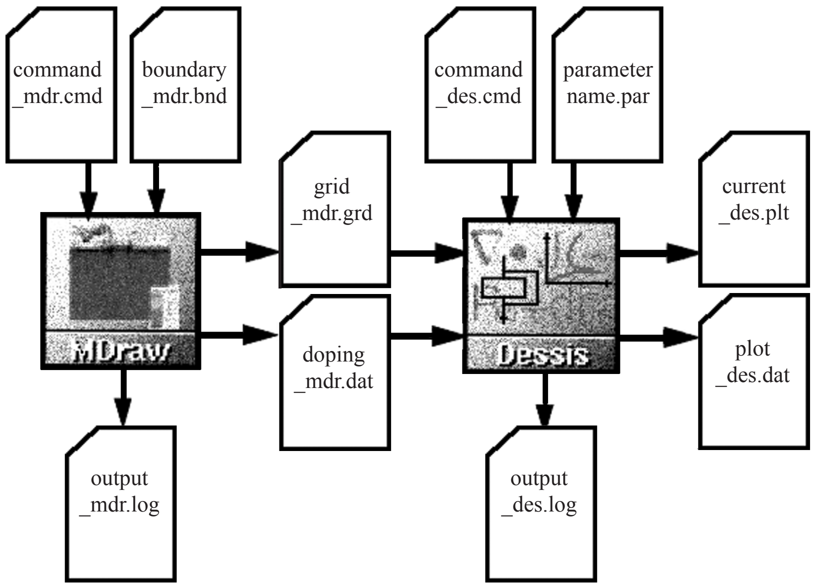


Figure 15.4 Typical design flow with DESSIS device simulation

1.2 Starting DESSIS

There are two ways to start DESSIS: from the command prompt or GENESISe.

1.2.1 Using the command prompt

DESSIS is driven by a command file and run by the command:

```
dessis <command_filename>
```

1.2.2 From GENESISe

DESSIS is launched automatically through the Scheduler when working inside GENESISe. Various options exist at start-up and are listed by using:

```
dessis -h
```

A DESSIS version is selected by using the option `-ver`:

```
dessis -ver 10.0.0 nmos_des.cmd
```

or:

```
dessis -ver 10.0.0 nmos_des
```

This commences a simulation using DESSIS Release 10.0. To use a previous version, the options `-rel` and `-ver` are required, for example, to start version 9.5.7:

```
dessis -rel 9.5 -ver 9.5.7 <command_file>
```

Appendix C on page 15.615 lists the DESSIS command options, which include:

`dessis -versions`	Checks which versions are in the installation path.
`dessis -P`	Extracts model parameter files (see Section 2.13.2 on page 15.91).
`dessis -L`	Extracts model parameter library (see Section 2.13.5 on page 15.92).
`dessis --parameter-names`	Prints parameter names that can be ramped (see Section 2.9.3.2 on page 15.60).

When DESSIS starts, the command file is checked for correct syntax, and the commands are executed in sequence. Character strings starting with `*` or `#` are ignored by DESSIS, so that these characters can be used to insert comments in the simulation command file.

NOTE GENESISe interprets `#` as a special marker for conditional statements (for example, `#if...`, `#elif...`, and `#endif...`).

1.3 Simulation examples

In the following sections, many of the DESSIS commands that are widely used are introduced in the context of a series of typical MOSFET device simulations. The examples are contained in the GENESISe projects, in the Examples Library, in the folder `GettingStarted/Dessis`.

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

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

1.4 Example: Simple MOSFET Id-Vg simulation

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

1.4.1 Input command file

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

An example of a complete command file (`des.cmd` in the GENESISe project `GettingStarted/Dessis/simple_Id-Vg`) is presented. Each statement section is explained individually.

```
File {* input files:
   Grid      = "nmos_mdr.grd"
   Doping    = "nmos_mdr.dat"
   * output files:
   Plot      = "n3_des.dat"
   Current   = "n3_des.plt"
   Output    = "n3_des.log"
}

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

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

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

Math {
   Extrapolate
   RelErrControl
}

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

Example_Library/GettingStarted/Dessis/simple_Id-Vg/des.cmd

1.4.2 File section

First, the input files that define the device structure and the output files for the simulation results must be specified. The device to be simulated is the one plotted in Figure 15.1 on page 15.3 and Figure 15.2 on page 15.4. The device is defined by the two files `nmos_mdr.grd` and `nmos_mdr.dat`:

```
File {
   * input files:
   Grid       = "nmos_mdr.grd"
   Doping     = "nmos_mdr.dat"
   * output files:
   Plot       = "n3_des.dat"
   Current    = "n3_des.plt"
   Output     = "n3_des.log"
}
```

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

`* input files:`	This is a comment line.
`Grid = "nmos_mdr.grd"`	This essential input file (default extension `.grd`) defines the mesh and various regions of the device structure, including contacts. DESSIS automatically determines the dimensionality of the problem from this file.
`Doping = "nmos_mdr.dat"`	This second essential input file (`.dat`) contains the doping profiles data for the device structure.
`* output files:`	This is a comment line.
`Plot = "n3_des.dat"`	This is the file name for the final spatial solution variables on the structure mesh (extension `_des.dat`).
`Current = "n3_des.plt"`	This is the file name for electrical output data (such as currents, voltages, charges at electrodes). Its standard extension is `_des.plt`.
`Output = "n3_des.log"`	This is an alternate file name for the output log or protocol file (default name `output_des.log`) that is automatically created whenever DESSIS is run. This file contains the redirected standard output, which is generated by DESSIS as it runs.

NOTE Only the root file names are necessary. DESSIS automatically appends the appropriate file name extensions, for example, `Plot="n3"` is sufficient.

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

1.4.3 Electrode section

Having loaded the device structure into DESSIS, it is necessary to specify which of the contacts are to be treated as electrodes. Electrodes in DESSIS are defined by electrical boundary conditions and contain no mesh. For example, in Figure 15.8 on page 15.15, the ‘polysilicon’ gate is empty; it is not a region.

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

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

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

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

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

`Name="string"`	Each electrode is specified by a case-sensitive name that must match exactly an existing contact name in the structure grid file. Only those contacts that are named in the `Electrode` section are included in the simulation.
`Voltage=0.0`	This defines a voltage boundary condition with an initial value. One or more boundary conditions must be defined for each electrode, and any value given to a boundary condition applies in the initial solution. In this example, the simulation commences with a 100 mV bias on the drain.
`Barrier=-0.55`	This is the metal–semiconductor work function difference or `barrier` value for a polysilicon electrode that is treated as a metal. This is defined, in general, as the difference between the metal Fermi level in the electrode and the intrinsic Fermi level in the semiconductor. This `barrier` value is consistent with n^+-polysilicon doping.

1.4.4 Physics section

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

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

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

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

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

`EffectiveIntrinsicDensity (BandGapNarrowing (OldSlotboom))`
This is the silicon band-gap narrowing model that determines the intrinsic carrier concentration.

1.4.5 Plot section

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

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

An extensive list of optional plot variables is in Table 15.14 on page 15.56. To save a variable as a vector, append `/Vector` to the keyword:

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

1.4.6 Math section

DESSIS solves the device equations (which are essentially a set of partial differential equations) self-consistently, on the discrete mesh, in an iterative fashion. For each iteration, an error is calculated and DESSIS attempts to converge on a solution that has an acceptably small error.

For this example, it is only necessary to define a few settings for the numeric solver. Other options, including selection of solver type and user definition of error criteria, are outlined in Section 2.10 on page 15.73.

```
Math {
   Extrapolate
   RelErrControl
}
```

`Extrapolate`	In quasistationary bias ramps, the initial guess for a given step is obtained by extrapolation from the solutions of the previous two steps (if they exist).
`RelErrControl`	Switches error control during iterations from using internal error parameters to more physically meaningful parameters (`ErrRef`) (see Section 2.10).

1.4.7 Solve section

The `Solve` section defines a sequence of solutions to be obtained by the solver. The drain has a fixed initial bias of 100 mV, and the source and substrate are at 0 V. To simulate the Id-Vg characteristic, it is necessary to ramp the gate bias from 0 V to 2 V, and obtain solutions at a number of points in-between. By default, the size of the step between solution points is determined by DESSIS internally, see Section 2.9.3 on page 15.58.

As the simulation proceeds, output data for each of the electrodes (currents, voltages, and charges) is saved to the current file `n3_des.plt` after each step and, therefore, the electrical characteristic is obtained. This can be plotted using INSPECT, as shown in Figure 15.5 and Figure 15.6 on page 15.13. The final 2D solution is saved in the plot file `n3_des.dat`, which is plotted in Figure 15.7 on page 15.14 and Figure 15.8 on page 15.15.

```
Solve {
   Poisson
   Coupled {Poisson Electron}

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

`Poisson`

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

`Coupled {Poisson Electron}`

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

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

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

1.4.8 Simulated Id-Vg characteristic

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

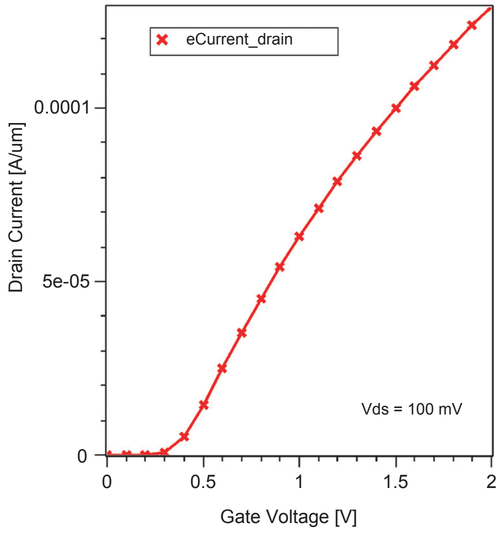


Figure 15.5 Id-Vgs characteristic of 0.18 µm n-channel MOSFET

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

In Figure 15.6, a log(Id)-lin(Vgs) plot shows the subthreshold characteristic.

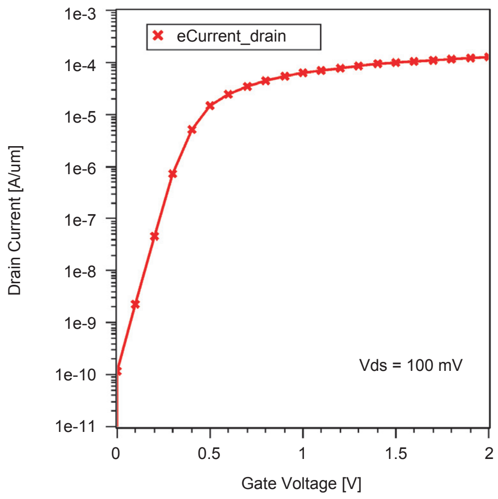


Figure 15.6 Id-Vgs characteristic of 0.18 µm n-channel MOSFET replotted on semi-log scale

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

```
Finished, because of...
Curve trace finished.
Writing plot 'n3_des.dat'... done.
```

The plot data in `n3_des.dat` can be analyzed using a graphics package such as Tecplot-ISE.

1.4.9 Analysis of 2D output data

The contents of file `n3_des.dat` is loaded into Tecplot-ISE, with the original grid file from MDRAW (`n1_mdr.grd`). The command is:

```
> tecplot_ise nl_mdr.grd n3_des.dat &
```

Figure 15.7 shows the default display, which is electrostatic potential.

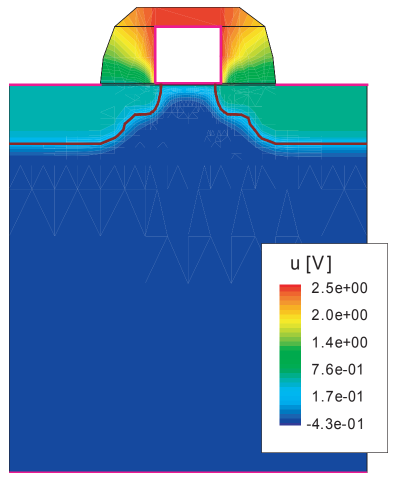


Figure 15.7 Default display of electrostatic potential and junctions

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

NOTE The potential at the gate electrode is 2.55 V (`InnerVoltage27`), which equals the applied bias minus the barrier potential. In the substrate, which is tied to 0 V, the ‘inner’ electrostatic potential is –0.4253 V, which corresponds to the position of the hole quasi-Fermi level relative to the intrinsic level.

Figure 15.8 is a magnification of the channel region, created by selecting the dataset `eDensity`. The inversion layer is clearly visible.

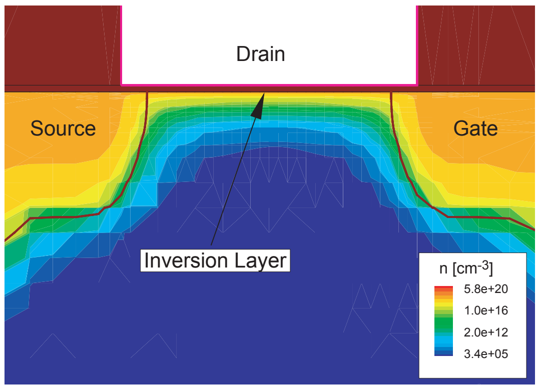


Figure 15.8 Magnification of channel region showing contours of electron concentration

Figure 15.9 shows the electron and hole densities along a vertical cutline in the center of the channel.

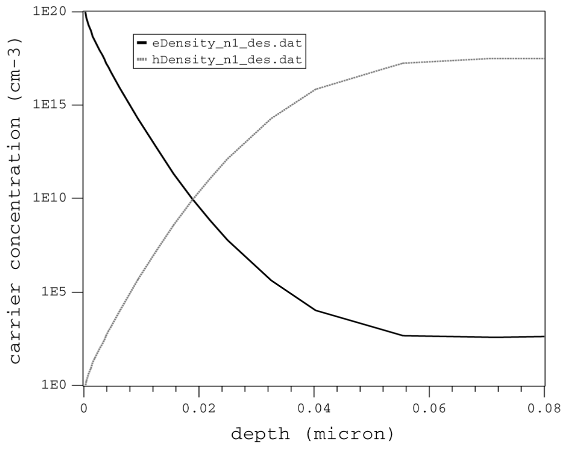


Figure 15.9 Vertical electron and hole profiles at center of channel

1.5 Example: Advanced hydrodynamic Id-Vd simulation

1.5.1 Input command file

```
#-----------------------------------------------------------------#
#- DESSIS input deck for
#-
#- Id=f(Vd) for Vd=0-10V  while Vg=[0.0V, 1.0V, 2.0V] and Vs=0V
#-
#- USING HYDRODYNAMIC MODEL & IMPACT IONIZATION - FAMILY OF CURVES
#-------------------------------------------------------------------#
File {
   Grid      = "@grid@"
   Doping    = "@doping@"
   Parameter = "mos"
   Plot      = "@dat@"
   Current   = "@plot@"
   Output    = "@log@"
}

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

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

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

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

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

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

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


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

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

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

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

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

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

1.5.2 File section

```
File {
   Grid      = "@grid@"
   Doping    = "@doping@"
   Parameter = "mos"
   Plot      = "@dat@"
   Current   = "@plot@"
   Output    = "@log@"
}
```

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

```
File {
   Grid      = "n1_mdr.grd"
   Doping    = "n1_mdr.dat"
   Parameter = "mos"
   Plot      = "n2_des.dat"
   Current   = "n2_des.plt"
   Output    = "n2_des.log"
}
```

`Grid`, `Doping`	The given file names relate to the appropriate node numbers in the GENESISe Family Tree. In most projects, `Grid` and `Doping` are generated by MDRAW or MESH and, therefore, have the characteristic names `nX_mdr.grd` and `nX_mdr.dat` from MDRAW, and `nX_msh.grd` and `nX_msh.dat` from MESH.
`Plot`, `Current`, `Output`	See Section 1.4.2 on page 15.9.
`Parameter = "mos"`	The optional input file `mos.par` contains user-defined values for model parameters (coefficients) (see Section 1.5.3). DESSIS adds the file extension `.par` automatically.

Main options

`(e/h)Lifetime = "<name>"`

Loads a lifetime profile contained in a data file. For 2D simulations, the profile is generated using MDRAW.

1.5.3 Parameter file

The parameter file contains user-defined values for model parameters (coefficients). The parameters in this file replace the values contained in a default parameter file `dessis.par`. All other parameter values remain equal to the defaults in `dessis.par`. Model coefficients can be specified separately for each region or material in the device structure (see Section 2.13.1 on page 15.91). Although this feature is intended for the simulation of heterostructure devices, it is useful in silicon devices as it allows materials, such as polysilicon, to have different mobilities.

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

```
dessis -P
```

A list of the parameter files is printed to the command window and into a file `dessis.par`, which is created in the working directory.

For other materials, such as gallium arsenide (GaAs) and silicon carbide (SiC), use:

```
dessis -P:GaAs
dessis -P:SiC
```

More options are described in Section 2.13.2 on page 15.91.

Listing of mos.par

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

Report in the protocol file n3_des.log

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

1.5.4 Electrode section

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

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

Main options

`Current=`	Defines a current boundary condition with initial value [A].
`Charge=`	Defines a floating electrode with a charge boundary condition and an initial charge value [C].
`Resistor=`	Defines a series resistance [Ω] (`AreaFactor`-dependent).
`eRecVelocity=`	Defines a recombination velocity [cm/s] at a contact for electrons (`hRecVelocity` for holes).
`Schottky=`	Defines an electrode as a Schottky contact. The attributes of such a contact are specified as `Barrier` and `(e)hRecVelocity`.
`AreaFactor=`	Specifies a multiplication factor for the current in or out of an electrode. It is preferable to define `AreaFactor` in the `Physics` section. In a 2D simulation, this can represent the size of the device in the third dimension (for example, gate width), which is 1 µm by default.
`Barrier=-0.55`	This is the metal–semiconductor work function difference or `barrier` value for an electrode that is treated as a metal. Defined, in general, as the difference between the metal Fermi level in the electrode and the intrinsic Fermi level in the semiconductor.

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

1.5.5 Physics section

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

`AreaFactor=0.4`	Specifies that the electrode currents and charges are multiplied by a factor of 0.4 (equivalent to a simulated gate width of 0.4 µm).
`Hydrodynamic(eTemperature)`	Selects hydrodynamic transport models for electrons only. Hole transport is modeled using drift-diffusion.
`Mobility( DopingDep Enormal`	See Section 1.4.4 on page 15.10.
`hHighFieldSaturation(GradQuasiFermi)`	Velocity saturation for holes. It uses the default model after Canali (based on Caughey–Thomas) (see Section 8.8 on page 15.193), and is driven by a field computed as the gradient of the hole quasi-Fermi level, which is the default.
`eHighFieldSaturation(CarrierTempDrive)`	Velocity saturation for electrons. It is based on an adaptation of the Canali model, driven by an effective field that is based on the electron temperature (kinetic energy) (see Section 8.8).
`Recombination(...)`	Defines the generation and recombination models.
`SRH(DopingDep)`	Shockley–Read–Hall recombination with doping-dependent lifetime (Scharfetter coefficients modified in the parameter file `mos.par`).
`eAvalanche(CarrierTempDrive)`	Avalanche multiplication for electrons is driven by an effective field computed from the local carrier temperature.
`hAvalanche(Eparallel)`	Avalanche multiplication for holes is driven by the component of the field that is parallel to the hole current flow. The default impact ionization model is from van Overstraeten–de Man (see Section 9.9 on page 15.213).

Main options

`Temperature`	Specifies the lattice temperature [K] (default 300 K).
`IncompleteIonization`	Incomplete ionization of individual species.

`GateCurrent(<model>)`	Selects a model for gate leakage or (dis)charging of floating gates (see Section 15.3.1 on page 15.293).
`Recombination( Band2Band )`	Simulates band-to-band tunneling.

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

1.5.6 Interface physics

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

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

Main options

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

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

1.5.7 Plot section

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

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

1.5.8 CurrentPlot section

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

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

In this example, the electrostatic potential and electron temperature are saved at three nodes corresponding to selected locations in the source (# 82), drain extensions (# 530), and the center of the body (# 1009). Node numbers are identified using the `VertexIndex` variable in Tecplot-ISE (see Tecplot-ISE, Section A.6 on page 5.29). Any number of nodes is allowed.

Main options

Any of the variables in Table 15.14 on page 15.56.

1.5.9 Math section

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

`NotDamped=50` Specifies the number of Newton iterations over which the right-hand side (RHS)-norm is allowed to increase. With the default of 1, the error is allowed to increase for one step only. It is recommended that `NotDamped > Iterations` is set to allow a simulation to continue despite the RHS-norm increasing.

`Iterations=20` Specifies the maximum number of Newton iterations allowed per bias step (default=50). If convergence is not achieved within this number of steps, for a quasistationary or transient simulation, the step size is reduced by the factor `decrement` (see Section 2.9.3 on page 15.58) and simulation continues.

`BreakCriteria {Current(Contact="drain" Absval=3e-4)}`
Break criteria are used to stop a simulation if a certain limit value is exceeded (see Section 2.10.5 on page 15.81). In this case, the simulation terminates when the drain current exceeds 3×10^{-4} A.

Example

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

`Method=` Selects the linear solver to be used in the coupled command.

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

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

1.5.10 Solve section

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

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

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

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

`Coupled { Poisson Electron }`

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

`Coupled { Poisson Electron Hole eTemperature }`

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

`Save (FilePrefix="vg0")`

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

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

Solve continued

```
# ramp gate and save solutions:

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

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

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

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

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

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

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

`{ Coupled { Poisson Electron Hole eTemperature } }`

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

`Save(FilePrefix="vg1") {`

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

Solve continued

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

`Load(FilePrefix="vg0")` Loads the solution file for zero gate bias.

`NewCurrentPrefix="vg0_"`

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

`Quasistationary` Implies that a series of quasi-static or steady state 'equilibrium' solutions are to be obtained.

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

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

`InitialStep=0.01 MaxStep=0.1 MinStep=0.0001 )`

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

`{ Coupled { Poisson Electron Hole eTemperature }`

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

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

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

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

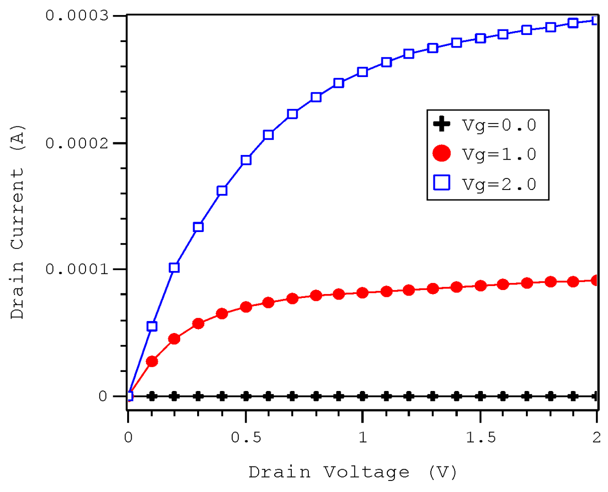


Figure 15.10 Family of drain output characteristics (Id-Vds) in drain bias range 0–2 V

Figure 15.10 shows the family of drain output characteristics (Id-Vds) in the drain bias range from 0 V to 2 V. The equal drain voltage steps of 0.1 V are suitable for SPICE parameter extraction. Figure 15.11 shows the Ids-Vds characteristics plotted with the electron temperature at the drain end of the channel over the full range of the simulations.

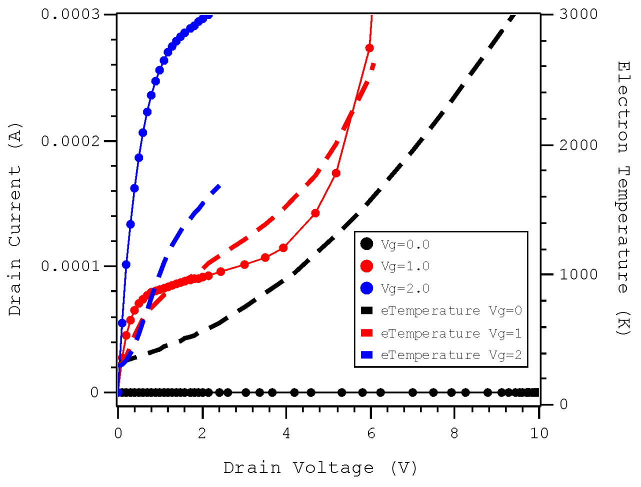


Figure 15.11 Ids-Vds characteristics plotted with electron temperature at drain end of channel over full range of simulations

1.5.11 Two-dimensional output data

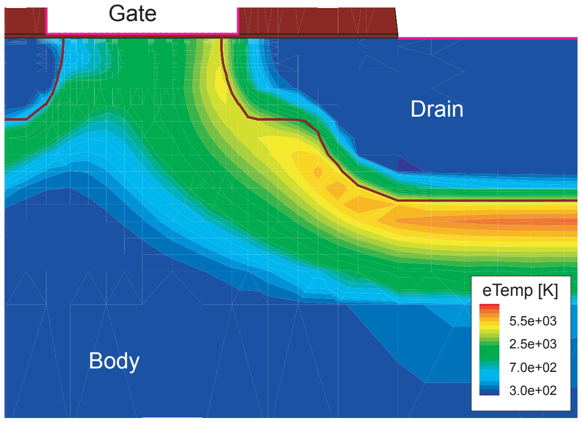


Figure 15.12 Contours of electron temperature computed at final solution (Vds=2.425 V, Id=30 mA)

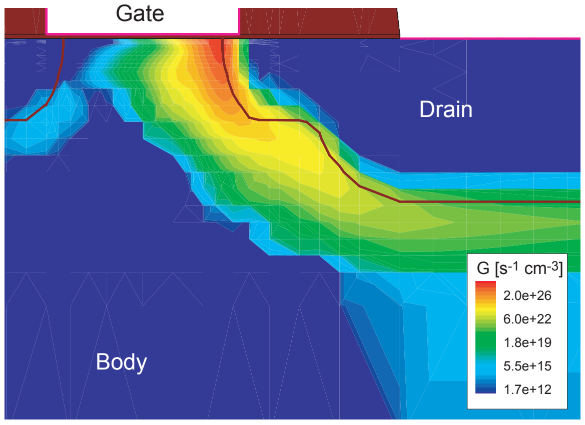


Figure 15.13 Contours of impact ionization rate at final solution

1.6 Example: Mixed-mode CMOS inverter simulation

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

In this example (Examples Library project `/GettingStarted/Dessis/Inverter`), a transient mixed-mode simulation is presented with two 2D physical devices; an n-channel and a p-channel MOSFET, combined with a capacitor and a voltage source to form a CMOS inverter circuit. MDRAW is used to create the 2D NMOSFET and PMOSFET devices. DESSIS computes the transient response of the inverter to a voltage signal, which codes a 010 binary sequence.

1.6.1 Input command file

```
#------------------------------------------------------------#
#- DESSIS input deck for a transient mixed-mode simulation of the
#- switching of an inverter build with a nMOSFET and a pMOSFET.
#------------------------------------------------------------#

Device NMOS {

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

Device PMOS{

   Electrode {
      { Name="source"    Voltage=0.0 Area=10 }
      { Name="drain"     Voltage=0.0 Area=10 }
      { Name="gate"      Voltage=0.0 Area=10  Barrier=0.55 }
      { Name="substrate" Voltage=0.0 Area=10 }
   }
   File {Grid    = "@grid:+1@"
      Doping     = "@doping:+1@"
      Plot       = "pmos"
      Current    = "pmos"
      Param      = "mos"
   }
   Physics {
      Mobility( DopingDep HighFieldSaturation Enormal )
      EffectiveIntrinsicDensity(BandGapNarrowing (OldSlotboom))
   }
}
   System {
      Vsource_pset v0 (n1 n0) { pwl = (0.0e+00 0.0
                                  1.0e-11 0.0
                                  1.5e-11 2.0
                                  10.0e-11 2.0
                                  10.5e-11 0.0
                                  20.0e-11 0.0)}
      NMOS nmos( "source"=n0 "drain"=n3 "gate"=n1 "substrate"=n0 )
      PMOS pmos( "source"=n2 "drain"=n3 "gate"=n1 "substrate"=n2 )
      Capacitor_pset c1 ( n3 n0 ){ capacitance = 3e-14 }
      Set (n0 = 0)
      Set (n2 = 2)
      Set (n3 = 2)
      Plot "nodes.plt" (time() n0 n1 n2 n3 )
   }
```

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

GettingStarted/Dessis/advanced_Inverter/des.cmd

1.6.2 Device section

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

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

For a mixed-mode simulation (see Section 3.3 on page 15.106), the physical devices are named `NMOS` and `PMOS`, and are defined in separate `Device` statements.

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

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

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

NOTE `Dessis{...}` and `Device{...}` are synonymous.

The equivalent section for the p-channel device is defined almost identically except that the source files for the p-channel structure are `@grid+1@` and `@doping+1@` (referring to the GENESISe node corresponding to the MDRAW split for the p-channel structure), and the plot and current files have the prefix `pmos`. Further, `Barrier=+0.55` for the p-channel MOSFET.

1.6.3 System section

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

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

   Capacitor_pset c1 ( n3 n0 ){ capacitance = 3e-14 }
   Set (n0 = 0)
   Set (n2 = 2)
   Set (n3 = 2)
   Plot "nodes.plt" (time() n0 n1 n2 n3 )
}
```

`Vsource_pset v0 (n1 n0)...`
A voltage source that generates a piecewise linear (`pwl`) voltage signal is connected between the input node (`n1`) and ground node (`n0`). The time–voltage sequence generates a low-high-low or 010 binary sequence over a 200 ps time period.

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

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

`PMOS pmos (...)` The PMOSFET is instantiated similarly.

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

`Set (n0 = 0)`

`Set (n2 = 2)`

`Set (n3 = 2)` The `Set` command defines the nodal voltages at the beginning of the simulation. These definitions are kept until an `Unset` command is specified. Node `n0` is tied to ground and `n2` is tied to the supply voltage 2 V. Node `n3` is set to 2 V.

1.6.4 File section

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

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

1.6.5 Plot section

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

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

1.6.6 Math section

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

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

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

1.6.7 Solve section

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

```
Solve {
   #- initial solution solved in steps
   Coupled { Poisson }
   Coupled { Poisson Electron Hole }
   Unset (n3)

   Transient (
      InitialTime=0 FinalTime=20e-11
      InitialStep=1e-12 MaxStep=1e-11 MinStep=1e-15
      Increment=1.3
   )
   {Coupled { nmos.poisson nmos.electron
            pmos.poisson pmos.hole }
   }
}
```

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

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

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

`{Coupled { nmos.poisson nmos.electron`
`pmos.poisson pmos.hole } }`
The `Coupled` statement illustrates how, in a mixed-mode environment, a specific set of equations can be selected. In this example, the electron continuity equation is solved in the NMOSFET. The hole continuity equation is solved for the PMOSFET. The contact and circuit equations are included by default.

1.6.8 Results of inverter transient simulation

The simulation results are plotted automatically using INSPECT, driven by the command file `pp3_ins.cmd`, which is created (or preprocessed) by GENESISe from the root command file `ins.cmd`. In Figure 15.14 on page 15.32, the transient response of the output voltage and drain current through the n-channel device is plotted, overlaying the input pulse.

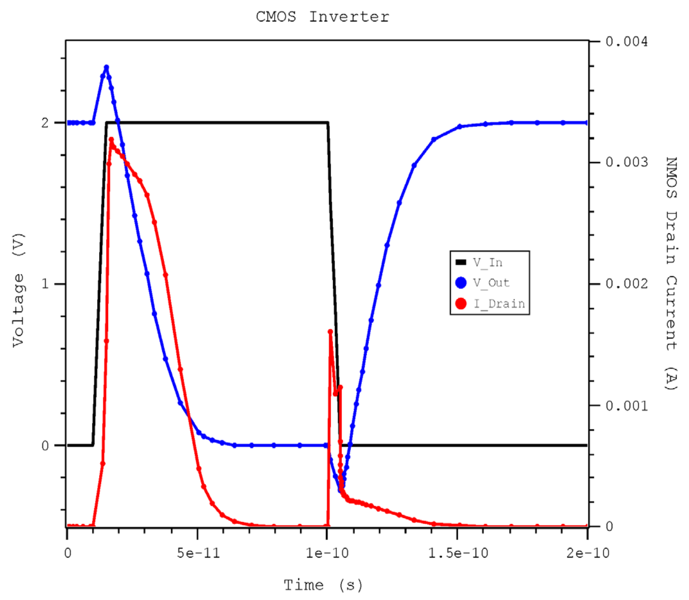


Figure 15.14 INSPECT output from GENESISe project (GettingStarted/Dessis/Inverter)

1.7 Example: Small-signal AC extraction

This example demonstrates how to perform an AC analysis simulation for an NMOSFET. In DESSIS, AC simulations are performed in mixed mode. In an AC simulation, DESSIS computes the complex (small signal) admittance Y matrix.

This matrix specifies the current response at a given node to a small voltage signal at another node:

$$i = Y \cdot u = A \cdot u + j \cdot \omega \cdot C \cdot u \qquad (15.1)$$

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

DESSIS output contains the components of the conductance matrix `A` and the capacitance matrix `C` (see Section 3.8.3 on page 15.117). The conductances and capacitances are used to construct the small signal equivalent circuit or to compute the other AC parameters, such as `H`, `Z`, and `S`.

1.7.1 Input command file

```
#-----------------------------------------------#
#- DESSIS input deck for
#- AC analysis at 1 MHz while Vg=-2 to 3V and Vd=2V
#-----------------------------------------------#
Device NMOS {

   Electrode {
      { Name="source"    Voltage=0.0 }
      { Name="drain"     Voltage=2.0 }
      { Name="gate"      Voltage=0.0 Barrier=-0.55 }
      { Name="substrate" Voltage=0.0 }
   }
   File {
      Grid    = "@grid@"
      Doping  = "@doping@"
      Current = "@plot@"
      Plot    = "@dat@"
      Param   = "mos"
   }
   Physics {
      Mobility ( DopingDep HighFieldSaturation Enormal )
      EffectiveIntrinsicDensity(BandGapNarrowing (OldSlotboom))
   }
   Plot {
      eDensity hDensity eCurrent hCurrent
      ElectricField eEparallel hEparallel
      eQuasiFermi hQuasiFermi
      Potential Doping SpaceCharge
      DonorConcentration AcceptorConcentration
   }
}
   Math {
      Extrapolate
      RelErrControl
      Notdamped=50
      Iterations=20
}
   File {
      Output    = "@log@"
      ACExtract = "@acplot@"
}
   System {
      NMOS trans (drain=d source=s gate=g substrate=b)
      Vsource_pset vd (d 0) {dc=2}
      Vsource_pset vs (s 0) {dc=0}
      Vsource_pset vg (g 0) {dc=0}
      Vsource_pset vb (b 0) {dc=0}
}
   Solve (
      #-a) zero solution
      Poisson
      Coupled { Poisson Electron Hole }

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

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

Example_Library/GettingStarted/Dessis/advanced_AC

1.7.2 Device section

```
Device NMOS {

   Electrode {
      { Name="source"   Voltage=0.0 }
      { Name="drain"    Voltage=2.0 }
      { Name="gate"     Voltage=0.0 Barrier=-0.55 }
      { Name="substrate" Voltage=0.0 }
   }
   File {
      Grid    = "@grid@"
      Doping  = "@doping@"
      Current = "@plot@"
      Plot    = "@dat@"
      Param   = "mos"
   }
   Physics {
      Mobility (DopingDep HighFieldSaturation Enormal)
      EffectiveIntrinsicDensity (BandGapNarrowing (OldSlotboom))
   }
   Plot {
      eDensity hDensity eCurrent hCurrent
      ElectricField eEparallel hEparallel
      eQuasiFermi hQuasiFermi
      Potential Doping SpaceCharge
      DonorConcentration AcceptorConcentration
   }
}
```

The AC analysis is performed in a mixed-mode environment (see Section 3.8.3 on page 15.117). In this environment, the physical device named `NMOS` is defined using the `Device` statement.

The `File` section inside `Device` includes all device-specific files, but cannot contain the `Output` identifier. This identifier is outside the `Device` statement in a separate global `File` section, with the file identifier for the data file, which contains the extracted AC results (see Section 1.7.3).

1.7.3 File section

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

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

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

1.7.4 System section

```
System {
   NMOS trans (drain=d source=s gate=g substrate=b)
   Vsource_pset vd (d 0) {dc=2}
   Vsource_pset vs (s 0) {dc=0}
```

```
      Vsource_pset vg (g 0) {dc=0}
      Vsource_pset vb (b 0) {dc=0}
   }
```

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

In this example, the `drain` voltage is set to 2 V directly when defining the voltage source instance `vd` (and also the `drain` voltage in the `Electrode` statement) in order to simplify and accelerate the simulation. Generally, it is more reliable to use a `Quasistationary` ramp in the `Solve` section to bias the device.

1.7.5 Solve section

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

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

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

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

```
#-c) ramp gate -2V to +3V : AC analysis
   Quasistationary (
      InitialStep=0.01 MaxStep=0.04 MinStep=1.e-5
      Goal { Parameter=vg.dc Voltage=3 }
      )
   { ACCoupled (
      StartFrequency=1e6 EndFrequency=1e6
      NumberOfPoints=1 Decade
```

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

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

`Node(d s g b)`	The AC analysis is performed between the circuit nodes `d`, `s`, `g`, and `b`. The conductance and capacitance matrices contain 16 elements each: `a(d,d)`, `c(d,d)`, `a(d,s)`, `c(d,s)`, ..., `a(b,b)`, `c(b,b)`.
`Exclude(vd vs vg vb)`	Excludes all voltage sources from the AC analysis.

1.7.6 Results of AC simulation

When DESSIS is run inside the GENESISe project `GettingStarted/Dessis/advanced_AC`, the simulation results are plotted automatically after completion by INSPECT, which is driven by the command file `pp3_ins.cmd`. The total small-signal gate capacitance `Cg=c(g,g)` is plotted against gate voltage. Two small-signal conductances are also plotted (see Figure 15.15), where `gm=a(d.g)` is the small-signal transconductance and `gd=a(d,d)` is the small-signal drain output conductance.

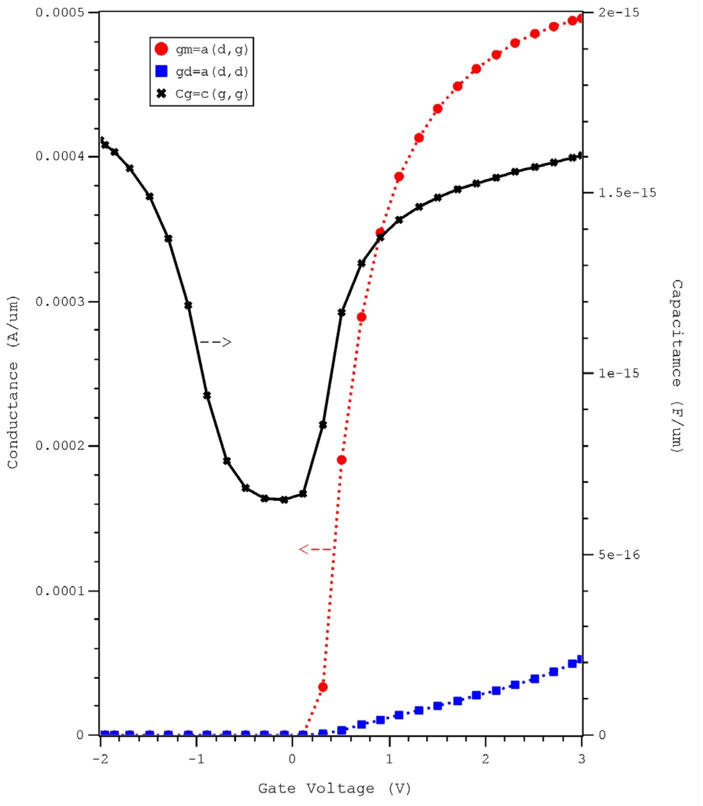
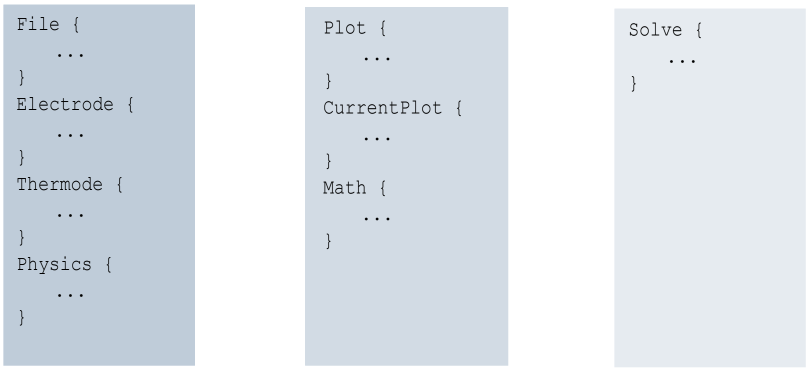


Figure 15.15 INSPECT output from GENESISe project AC simulation

CHAPTER 2 Basic DESSIS

2.1 Overview

The DESSIS input file is divided into sections that are defined by a keyword and braces (see Figure 15.16). A device is defined by the `File`, `Electrode`, `Thermode`, and `Physics` sections. The solve methods are defined by the `Math` and `Solve` sections. Two sections are used in mixed-mode circuit and device simulation, see Chapter 3 on page 15.101. This part of the manual concentrates on the input to define single device simulations.



```
File {
   ...
}
Electrode {
   ...
}
Thermode {
   ...
}
Physics {
   ...
}
```

```
Plot {
   ...
}
CurrentPlot {
   ...
}
Math {
   ...
}
```

```
Solve {
   ...
}
```

Figure 15.16 Different sections of a DESSIS input file

2.1.1 Specifying the device

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

2.1.2 Defining the output

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

2.1.3 Specifying the simulation

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

2.2 File section

File names for a simulation are specified in the `File` section. Each command uses a predefined file extension. Therefore, file names are given without extensions. Table 15.1 lists the input file options.

Table 15.1 Device input files

File command	Description	GENESISe reference
`Grid`	Device geometry and mesh description file that is supplied by a mesh generator (`.grd`).	`@grid@`
`Doping`	Device doping file that describes impurity concentration. It is supplied by a mesh generator (`.dat`) and matches the grid in the `.grd` file.	`@doping@`
`LifeTime`	Loads lifetime profiles for electrons and holes (`.dat`).	
`Load`	Loads and continues the simulation with old device results (`.sav`).	
`Parameters`	Reads and uses model parameters from the file for the device (`.par`). A template parameter file is generated using the DESSIS option `-P` (see Section 2.5.3 on page 15.47). The template parameter file `dessis.par` is ASCII formatted and can be renamed, modified, and loaded in order to adjust parameters of the physical models.	`@parameter@`

In most cases, a device can be specified using only the `Grid` and `Doping` files. Where a simulation depends on previous results, the `Load` command loads a previously computed solution. The `Parameters` command is used to change the standard model parameters for a device. The outputs of a device simulation can be saved for reuse (command `Save`) or plotting (command `Plot`). The voltage, charge, and current values at the electrodes can be logged (command `Current`). The `Plot` option stores a default set of variables and any variables specified by the user in the `Plot` section. The `Save` option saves only data that is necessary to restart the simulation after a `Load` command. These differences are summarized in Table 15.2.

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

Other device-related files are `OpticalGenerationFile`, (see Section 13.4 on page 15.266) and `Piezo` (see Chapter 22 on page 15.353).

Table 15.2 Device output files

File command	Description	GENESISe reference
`Save`	Stores a solution for future use as the initial solution after a `Load` command (`_des.sav`).	
`Plot`	Stores device physical output data for visualization with a graphics package (`_des.dat`).	`@dat@`
`Current`	Stores device currents, voltages, charges, and temperatures (and times, in transient simulations) for subsequent plotting and analysis (`_des.plt`).	`@plot@`
`ACExtract`	Specifies the file in which the results of small signal AC analysis are stored (`_ac_des.plt`).	`@acplot@`

Table 15.2 Device output files

File command	Description	GENESISe reference
`Output`	The general output file command. This file is a plain text compilation of the run-time output created by the simulation (`_des.log`).	`@log@`
`NonLocalPlot`	Specifies the file for the output of data defined on nonlocal line meshes (see Section 2.10.7.3 on page 15.85).	
`NewtonPlot`	Specifies a file name for the output of convergence-related data (see Section 2.10.8.2 on page 15.88).	

NOTE The GENESISe references in Table 15.1 and Table 15.2 on page 15.38 are valid using the GENESISe default `tooldb.tcl` (see GENESISe, Section 3.5.3 on page 1.91).

2.3 Electrode section

The `Electrode` section defines the electrical contacts of a device.

2.3.1 Command syntax

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

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

By default, contacts are Ohmic or gate contacts (see Section 4.5.1 on page 15.138). Table 15.3 lists the additional options that can be specified for each electrode.

Table 15.3 Electrode options

Keyword	Description

`Resist = <float>`
`DistResist = <float>`
	Defines a resistive contact (Ω *µm for 2D) or a distributed contact resistance at a contact (Ω *cm^2).

`DistResist = SchottkyResist`	Activates the doping dependent Schottky contact resistance model (see Section 4.5.1.5 on page 15.141).
`Current = <float>`	Defines a current boundary condition (A/µm for 2D). Voltage is only specified when initialization is necessary.
`Barrier = <float>`	Specifies a barrier voltage for gates [V].
`Workfunction = <float>` or `Material = "name"`	Specifies a work function [eV] for a gate or Schottky electrode (see Section 5.2.1 on page 15.151).

Table 15.3 Electrode options

Keyword	Description
`Charge = <float>`	Defines a floating electrode with a given charge [C]. Voltage must not be specified (see Section 4.5.1 on page 15.138 and Section 4.5.1.7 on page 15.144).
`AreaFactor = <float>`	Specifies a factor by which the electrode current is multiplied.
`Schottky`	Defines a Schottky contact. Attributes are `Barrier/Workfunction` and `(e/h)RecVelocity` (see Section 4.5.1.3 on page 15.140).
`eRecVelocity, hRecVelocity = <float>`	Defines recombination velocities for electrons and holes [cm/s] (defaults are 2.573 x 10^6 and 1.93 x 10^6 cm/s, respectively).
`FGcap=(value=<float> name="ContactName")`	If the electrode is floating, this specifies the value of an additional capacitance (F/µm for 2D) coupling to any other electrode given by `"ContactName"` (see Section 4.5.1.6 on page 15.143).

The parameters `Charge` and `Current` are conceptually different from the other parameters in that they determine the boundary condition type for an electrode (which is related to an experimental setup), while the other parameters describe physical properties of the electrodes (that relate to the device itself). By default, electrodes have a voltage boundary condition type. The keyword `Current` changes the boundary to current type, and the keyword `Charge` changes it to charge type. It is possible to change the boundary condition for an electrode during the simulation from current type to voltage type (see Section 2.9.3 on page 15.58 and Section 2.9.10 on page 15.73).

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

Units of the parameters `Resist`, `Current`, and `FGcap` are specified for 2D cases. The influence of `AreaFactor` on these values is based on a rule that the voltage drop across the resistor does not change for different values of `AreaFactor`. Therefore, if the measured values R [Ω], I [A], and C [F] are known, then the electrode values should be computed as follows: `Resist=`R*`AreaFactor`, `Current=`I/`AreaFactor`, and `FGcap=`C/`AreaFactor`. For example, if a 5 Ω contact resistance is experimentally measured for a 10 µm wide contact, `AreaFactor=10` and `Resist=50` (5x10 Ω *µm) is used. However, to simulate a half contact, `AreaFactor=20` and `Resist=100` must be used.

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

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

2.3.2 Work function and material specifications for contacts

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

The work function is specified using the syntax `Workfunction = <value>` [eV] in the `Electrode` statement, for example:

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

To use a default work function of a material (or from the parameter file), the material name can be specified: `Material = "Name"`. For electrodes such as a MOSFET gate, it is useful to specify both the semiconductor material and doping concentration (for example, for polysilicon gates). In this case, the syntax is `Material = "Name"(N = <value>)` for n-type impurities, and `Material = "Name"(P = <value>)` for p-type impurities, for example:

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

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

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

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

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

Table 15.4 Various electrode declarations

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

Table 15.4 Various electrode declarations

Description	Command statement
In the above example, the source assumes these voltages during a transient simulation: 0 V at 0 s, 0.2 V at 1 µs, and 0.5 V at 2 µs.	
Time-dependent voltage drive	`{ name = "source" voltage = 0 voltage = (5 at 0, 10 at 1e-6) }`
In the above example, both the initial voltage (0 V) and time-dependent values (5 V at 0 s, 10 V at 1 ms) have been specified for the source. This combination is useful where an initial quasistationary command ramps the source voltage from 0 V to 5 V, before the source voltage increases to 10 V during a subsequent transient analysis.	

2.4 Thermode section

The `Thermode` section defines the thermal contacts of a device.

2.4.1 Command syntax

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

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

A complete `Thermode` section can be:

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

Table 15.5 lists the options available for the `Thermode` section.

Table 15.5 Thermode options

Keyword	Description
`SurfaceResistance`	Defines a contact thermal resistivity [cm^{2}K/W].
`SurfaceConductance`	The alternative inverse value of `SurfaceResistance` (thermal conductivity).
`Power = <float>`	Defines a heat flux boundary condition [W/cm^2]. Temperature is specified when initialization is necessary.
`AreaFactor = <float>`	Specifies a multiplicative factor that multiplies the area of the thermode by a given amount.

Table 15.6 Various thermode declarations

Command statement	Description
`{name = "surface" Temperature = 310 SurfaceResistance = 0.1}`	This is a thermal resistive boundary condition with 0.1 cm^2 K/W thermal resistance, which is specified at the thermode 'surface.'
`{name = 0 Temperature = 300 Power = 1e6}`	Heat flux boundary condition.

Table 15.6 Various thermode declarations

Command statement	Description
`{name = 1  Temperature = 300  Power=1e5 power = (1e5 at 0, 1e6 at 1e-4, 1e3 at 2e-4)}`	Thermode with time-dependent heat flux boundary condition.

2.5 Physics section

The `Physics` section is used to select a global set of models that are used to simulate a device. Table 15.7 on page 15.44 lists the principal options that are available and Table 15.8 on page 15.45 lists other options. Other less frequently used parameters are presented in Chapter 4 on page 15.127.

The two types of option in the `Physics` section are:

- Model selection commands (for example, `Mobility`, `Recombination`).
- Modifiers of parameters for global models (for example, `Thermodynamic`, `Hydrodynamic`).

2.5.1 Example: Possible input parameters

This example illustrates many of the possible input parameters for the `Physics` section:

```
Physics {
   Temperature=300
   Charge(Concentration=0)
   # bandgap narrowing on:
   EffectiveIntrinsicDensity (BandGapNarrowing (BennettWilson ))
   Mobility (
      DopingDependence
      # default model:
      CarrierCarrierScattering(ConwellWeisskopf)
      HighFieldSaturation # default GradQuasiFermi
      Enormal
   )
   Recombination (
   # no trap-assisted tunneling:
   SRH ( DopingDependence )
   Auger
   TrapAssistedAuger
   # using non-default driving force, but default model:
   Avalanche(vanOverstraeten Eparallel)
   Band2Band
   )
   AlphaParticle (
      Energy=5e6 Time=0 # default time
      Direction = (1,2)
      Location  = (5,0)
   )
}
```

2.5.2 Main and additional options

Table 15.7 Principal options for Physics section

Keyword	Description
`AreaFactor = <float>`	Specifies a global area factor for the device, which is equivalent to specifying the area factor in each electrode and thermode.
`Charge(Concentration=<float>)`	Specifies the oxide charge density [q/cm^3] where q is the elementary charge. Default is 0 q/cm^3.
`EffectiveIntrinsicDensity (BandGapNarrowing (<models>)`	Selects the model for the effective intrinsic density, $n_{i,eff}$, and switches off band-gap narrowing if required (see Section 5.2 on page 15.151).

`Hydrodynamic`
`Hydrodynamic (<carrier>)`
	Switches on the hydrodynamic transport model (see Section 4.2.4 on page 15.130).
	Switches on the hydrodynamic transport model for a specified carrier only (electron or hole).

`Mobility(<models>)`	Selects the mobility model. The default is the temperature-dependent constant mobility model. Doping dependence, saturation, normal field effects, and carrier–carrier scattering are switched on with extra parameters (see Chapter 8 on page 15.175).
`Recombination (<models>)`	Selects the generation–recombination model. The default is that all generation–recombination models are switched off. Shockley–Read–Hall (SRH) recombination, coupled defect level (CDL) recombination, Auger recombination, trap-assisted Auger (TAA) recombination, band-to-band tunneling, and avalanche generation are switched on with extra parameters (see Chapter 9 on page 15.201).
`Temperature = <float>`	Specifies the device lattice temperature [K]. Default is 300 K.
`Thermodynamic`	Switches on the thermodynamic transport model (see Section 4.2.3 on page 15.128).
`Traps (<options>)`	Switches on bulk distributed traps for transient or steady state simulations (see Chapter 10 on page 15.225).

NOTE The hydrodynamic transport model can only be activated for the whole device. A region-specific or material-specific activation is not possible, and the `Hydrodynamic` keyword(s) is ignored in any `Physics` section other than the global one.

NOTE For the `Traps` and `Charge` statements, the syntax allows specification of both statements inside any `Physics` section. However, `Traps` applies *only* to semiconductor regions and is ignored in insulators. `Charge` applies only to insulators.

Table 15.8 lists modifiers for models in the `Physics` section. Additional keywords may be found in Chapter 4 on page 15.127.

Table 15.8 Additional options for Physics section

Option	Description
`AlphaParticle (<options>)`	Switches on generation of electron–hole pairs by an impinging alpha particle for a transient SEU simulation (see Section 14.1 on page 15.283).
`Amorphous (<options>)`	Switches on bulk distributed traps, typical of amorphous and polycrystalline silicon (see Chapter 10 on page 15.225).
`AnalyticTEP`	Switches on analytic dependencies of thermoelectric power (default uses experimental values) (see Section 24.5 on page 15.367).
`Fermi`	Switches on Fermi–Dirac statistics, which can only be activated for the whole device. A region-specific or material-specific activation is not possible (see Section 4.4 on page 15.137).
`GateCurrent(<model>)`	Selects a gate current model for memory cell and leakage problems. By default, all models are off (see Section 15.3.1 on page 15.293).
`HeavyIon (<options>)`	Switches on generation of electron–hole pairs by an impinging heavy ion for a transient SEU simulation (see Section 14.2 on page 15.284).
`IncompleteIonization`	Switches on incomplete ionization (see Chapter 6 on page 15.161).
`MagneticField = (<float>,<float>,<float>)`	Specifies magnetic field [Tesla] (see Chapter 23 on page 15.363).
`MoleFraction`	Specifies mole fraction distribution for compound semiconductors (see Section 18.4 on page 15.323).
`Noise (<options>)`	Switches on noise sources (see Chapter 15 on page 15.291).
`OptBeam (<options>)`	Used for simple simulation of optical generation (see Section 13.1 on page 15.243).
`Piezo (<specifications>)`	Switches on piezoresistive effects (see Section 22.5 on page 15.359).
`[e | h]QCvanDort`	Activates the van Dort quantum correction model (see Section 7.2 on page 15.166).
`Radiation (<options>)`	Switches on the radiation model (see Chapter 12 on page 15.241).
`RecGenHeat`	In hydrodynamic mode, switches on heat sources and sinks due to recombination and generation processes (see Section 4.2.4 on page 15.130).
`Schroedinger`	Switches on the 1D Schrödinger solver (see Section 7.3 on page 15.167).

2.5.2.1 Effective intrinsic density keywords

Table 15.9 lists the parameters for the effective intrinsic density models.

Table 15.9 Keywords for effective intrinsic density models

`EffectiveIntrinsicDensity (BandGapNarrowing (<model type>)):`	`<model type>`
	`BennettWilson`
	`oldSlotboom`
	`Slotboom`
	`delAlamo`

2.5.2.2 Mobility keywords

Table 15.10 lists the optional keywords for mobility models. More than one mobility model can be activated simultaneously.

NOTE An old option with electric field normal to current density vector is available and is activated by the keyword `ToCurrentEnormal`, but it is less robust.

Table 15.10 Keywords for mobility models

`DopingDependence` Switches on the doping dependence of the mobility.	

`HighFieldSaturation(<driving force>)`
Switches on the high field saturation model of the mobility. An optional parameter defines driving force.
The default is `GradQuasiFermi`.
 	
Optional:
`[e|h]HighFieldSaturation (<driving force>)`
Allow different driving forces to be specified for electron and hole high field saturation mobility models.
	`<driving force>`

	`GradQuasiFermi`
	`Eparallel`
	`CarrierTempDrive`
	`CarrierTempDriveBasic`
	`CarrierTempDriveME`
`NormalElectricField` or `Enormal` Switches on dependence of mobility to normal electric field. The keywords `ToInterfaceEnormal` and `Enormal` are synonymous, that is, in the mobility degradation model, electric field normal to silicon–oxide interface is used.	
`CarrierCarrierScattering (<model type>)` Switches on carrier–carrier scattering mobility model. An optional model type can be specified. The default is `ConwellWeisskopf`.	`<model type>`
	`ConwellWeisskopf`
	`BrooksHerring`
`PhuMob (<donor species>)` Switches on the Philips unified mobility model. An optional donor species can be specified. The default is `Arsenic`.	`<donor species>`
	`Arsenic`
	`Phosphorus`

2.5.2.3 Generation–recombination keywords

By default, all generation–recombination models are switched off. They must be specified as options of the keyword `Recombination`. Table 15.11 lists available models.

It is possible to specify different models or driving forces for the electron and hole avalanche generation terms. This flexibility is achieved by using the keywords `eAvalanche` or `hAvalanche` with the appropriate options.

Table 15.11 Keywords for generation–recombination models

`SRH(`*`<model type>`*`)` Switches on SRH recombination. Optional model types can be specified (see Section 9.1 on page 15.201).	*`<model type>`*	
	`DopingDependence`	
	`Tunneling` (see Section 9.2 on page 15.204)	
	`TempDependence`	
	`ExpTempDependence`	
`CDL(`*`<model type>`*`)` Switches on the CDL recombination or trap-assisted tunneling (see Section 9.5 on page 15.209). Optional model types can be specified.	*`<model type>`*	
	`DopingDependence`	
	`Tunneling`	
	`TempDependence`	
	`ExpTempDependence`	
`[+|-]Radiative(`*`<model type>`*`)` Switches on radiative recombination (see Section 9.6 on page 15.211).		
`Auger` Switches on Auger recombination (see Section 9.7 on page 15.212).		
`TrapAssistedAuger` Switches on TAA recombination (see Section 9.8 on page 15.213).		
`Band2Band` Switches on band-to-band tunneling generation (see Section 9.11 on page 15.221 and Section 16.4 on page 15.306).		
`[e|h]Avalanche(`*`<model type> <driving force>`*`)` Switches on avalanche generation (see Section 9.9 on page 15.213). The default model type is `VanOverstraeten`. The default driving force is `GradQuasiFermi`.	*`<model type>`*	*`<driving force>`*
	`VanOverstraeten`	`GradQuasiFermi`
	`Okuto`	`Eparallel`
	`Lackner`	`CarrierTempDrive`

2.5.3 Region-specific and material-specific physics

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

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

or:

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

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

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

or:

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

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

A `Physics` section without any region or material specifications is considered the default section. The hierarchy of region or material `Physics` and `Math` specifications is presented in Section 2.5.4.

NOTE Region names can be defined using MDRAW (see MDRAW, Section 2.4.6 on page 11.10).

Regionwise or materialwise specification is not allowed for these models:

```
AnalyticTEP      Fermi        Hydrodynamic  MagneticField    Piezo
QCVanDort        RecGenHeat   SiC           Temperature      Thermodynamic
```

See Section 2.13 on page 15.91.

2.5.4 Hierarchy of physical model specifications

To avoid confusion, it is important to understand the hierarchy of region and material `Physics` and `Math` specifications. For specification of the `Physics` section, consider:

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

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

For example:

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

If the `"Emitter"` region is made of `GaAs`, the models in this region are `<Default models>` and `<Emitter models>`, that is, `<GaAs models>` is not added.

For some models, the model specification and numeric values of the parameters are defined in the `Physics` sections. Examples of such models are `Traps` and the `MoleFraction` specifications. For these models, the specifications in region or material `Physics` sections overwrite previously defined values of the corresponding parameters.

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

2.5.5 Physics at interfaces

A special set of models can be activated at the interface between two different materials or two different regions. Table 15.12 lists the interface models supported by DESSIS.

Table 15.12 Interface Physics options

`Recombination(<model>)` Activates surface SRH recombination or tunneling model at a heterointerface.	`<model>`	
	`surfaceSRH`	
	`[e|h]BarrierTunneling` (Section 16.4 on page 15.306)	
`GateCurrent(<model>)` All gate current models can be applied to selected interfaces.	`<model>`	
	`Fowler` (Section 16.2 on page 15.300)	
	`DirectTunneling` (Section 16.3 on page 15.302)	
	`[e/h]Lucky` (Section 17.2 on page 15.318)	
	`[e/h]Fiegna` (Section 17.3 on page 15.319)	
`[e/h]Thermionic` Activates the thermionic emission model at an interface.	(Section 18.10 on page 15.330)	
`Traps(<options>)` Interface traps are specified in the same way as bulk traps.	(Section 2.5.5.3 on page 15.50)	
`Charge(<option><parameters>)` Fixed charges can be specified at selected interfaces. The default option is `Uniform`.	`<option>`	`<parameters>`
	`Uniform`	`SurfConc=<float>` [1/cm^{-2}]
	`Gaussian`	`SpaceMid=(<float>,<float> [,<float>])` [µm]
	(Section 2.5.5.4 on page 15.51)	`SpaceSig=(<float>,<float> [,<float>])` [µm]

As physical phenomena at an interface are not the same as in the bulk of a device, other physical models are not allowed inside the `Interface Physics` statements. It is not possible to define any mobility models or band-gap narrowing at interfaces.

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

2.5.5.1 Interface model syntax

Interface models are specified in the input file in the same way as `Physics` statements. Their respective parameters are accessible in the parameter file. The syntax for specification of an interface model is:

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

or:

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

These `Interface Physics` statements replace and enhance the `InterfaceCondition` statement that existed in previous DESSIS versions. The following is an example illustrating the specification of fixed charges at the interface between the materials oxide and aluminum gallium arsenide (AlGaAs):

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

2.5.5.2 Recombination and gate current

The use of these models is detailed in the `Physics` sections of Section 9.4 on page 15.208 and Section 15.3.1 on page 15.293, respectively.

2.5.5.3 Interface traps

The `Traps` statement inside a material `Physics` section has the same syntax as the `Traps` statement in a regular (bulk) `Physics` section (see Chapter 10 on page 15.225). However, for interfaces, the sheet concentration (`Conc`) is given in units of cm^{-2}.

NOTE Wherever a contact exists at a specified region interface, DESSIS does not recognize the interface traps within the bounds of the contact because the contact itself constitutes a region and effectively overwrites the interface between the two ‘material’ regions. This is true even if the contact is not declared in the `Electrode` statement.

2.5.5.4 Interface charge

Fixed charges at interfaces are specified with either Gaussian or uniform distributions:

```
Charge([Uniform | Gaussian]
   SurfConcentration = <number>    # [cm-2]
      SpaceMid = (<x,y,[z]>)       # [mm]
      SpaceSig = (<x,y,[z]>) )     # [mm]
```

NOTE The default is `Uniform`. For uniform distributions, the variable `SurfConcentration` (synonym `Conc`) specifies the uniform charge concentration. The optional keywords `SpaceMid` and `SpaceSig` allow the charges to be restricted to the part of the interface that intersects a box with its center at the point `SpaceMid`. The variable `SpaceSig` contains the distances in the x and y (and z, for 3D) directions from the center to the sides of the box.

For Gaussian distributions, the variable `SurfConcentration` specifies the maximum charge concentration. The vector `SpaceMid` points to the peak of the Gaussian, and `SpaceSig` denotes the standard deviations in the x and y (and z, for 3D) directions. The charge distribution at the interface is given by the intersection of the Gaussian with the interface.

It is possible to have several specifications sections in a single `Charge` statement:

```
Charge((<spec_1>) (<spec_2>) ... (<spec_n>))
```

For example, the syntax allows the specification of a piecewise constant interface charge distribution:

```
Charge ((Uniform Conc=-1.e12 SpaceMid=(0.02,0) SpaceSig=(0.01,0)
      (Gaussian Conc=2.e12 SpaceMid=(0.04,0) SpaceSig=(0.01,0)))
```

2.5.5.5 Interface model parameters

If a model is defined in an `Interface Physics` section, the parameters of that model must be specified in the parameter file in a section corresponding to the same interface. For example, for the statement:

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

the corresponding model parameters must be defined in the DESSIS parameter file:

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

This parameter definition also applies to:

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

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

2.5.6 Physics at electrodes

An electrode-specific `Physics` section can be defined for a Schottky electrode with appropriate keywords for the barrier tunneling model (see Section 16.4 on page 15.306), for example:

```
Physics(Electrode="Gate"){
   Recombination(BarrierTunneling)
}
```

To activate the barrier tunneling model, the electrode `Gate` must also be specified as `Schottky` in either the `Electrode` section (see Section 4.5.1.3 on page 15.140) or the same `Physics` section as follows:

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

2.6 Plot section

The `Plot` section specifies the data that is saved by the `Plot` command in the `File` section. Each data field is defined with a keyword. Appendix E on page 15.619 lists the keywords.

NOTE To save these variables as a vector, add `/Vector` to the corresponding keyword.

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

2.7 CurrentPlot section

This section is used to include selected mesh data into the current plot file (`.plt`). The same variables can be selected as in the `Plot` section (see Appendix E on page 15.619).

Data can be plotted according to node numbers or coordinates. The node numbers are obtained by probing the mesh using Tecplot-ISE (see Tecplot-ISE, Section A.6 on page 5.29).

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

Furthermore, it is possible to output averages, and maximum and minimum of quantities over specified domains. To do this, specify the keyword `Average`, `Maximum`, or `Minimum`, respectively, followed by the specification of the domain in parentheses. A domain specification consists of any number of the following:

- Region specification: `Region=<`*`regionname`*`>`
- Material specification: `Material=<`*`materialname`*`>`
- Any of the keywords `Semiconductor`, `Insulator`, and `Everywhere`, which match all semiconductor regions, all insulator regions, or the entire device, respectively

The average (or maximum or minimum) is applied to all of the specified parts of the device. Multiple specifications of the same part of the device are insignificant. In addition, `Name=<plotname>` is used to specify a name under which the average (or maximum or minimum) is written to the `.plt` file. (By default, the name is automatically obtained from a concatenation of the names in the domain specification, which yields impractically long names for complicated specifications.)

Parameters from the DESSIS parameter file can also be added to the current plot file. The general specification looks like:

```
[ Material = <material> | MaterialInterface = <interface> |
   Region = <region> | RegionInterface = <interface> ]
Model = <model> Parameter = <parameter>
```

Specifying the location (material, material interface, region, or region interface) is optional. However, the model name and parameter name must always be present. Section 2.9.3.2 on page 15.60 describes how model names and parameter names can be determined. Finally, DESSIS also provides a current plot PMI (see Section 33.28 on page 15.600).

NOTE Do not confuse the `CurrentPlot` section with the `CurrentPlot` statement in the `Solve` section introduced in Section 2.9.9 on page 15.71.

2.7.1 Example: Node numbers

Consider this `device` declaration in a DESSIS input file:

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

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

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

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

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

2.7.2 Example: Mixed mode

In mixed-mode simulations, the `CurrentPlot` section can appear in the body of a physical device within the `System` section (it is also possible to have a global `CurrentPlot` section), for example:

```
System {
   Set (gnd = 0)
   CAP Cm (Top=node2 Bot=gnd) { CurrentPlot { Potential (7, 8, 9) } }
   ...
}
```

2.7.3 Example: Advanced options

This example is a 2D device that uses the more advanced `CurrentPlot` features. It generates 11 curves:

```
CurrentPlot {
   eDensity (0 1)  * plot electron density at nodes 0 and 1
   hDensity((0 1)) * hole density at position (0um, 1um)
   ElectricField/Vector((0 1)) * Electric Field Vector
   Potential (
      (0.1 -0.2)  * coordinates need not be integers
      Average(Region="Channel")  * average over a region
      Average(Everywhere)        * average over entire device
      Maximum(Material="Oxide")  * Maximum in a material
      Maximum(Semiconductor)     * in all semiconductors
      * material specification is redundant, therefore the same as above will be plotted:
      Maximum(Semiconductor Material="Silicon")
      * minimum in a material and a region, output under the name "x":
      Minimum(Name="x" Material="Oxide" Region="Channel")
      * minimum in a region and all insulator regions:
      Minimum(Region="Channel" Insulator)
   )
}
```

2.7.4 Example: Physical parameter values

The following example adds five parameters to the current plot file:

```
CurrentPlot {
   Model = DeviceTemperature   Parameter = "Temperature"
   Material = Silicon   Model = Epsilon   Parameter = epsilon
   MaterialInterface = "AlGaAs/InGaAs"   Model = "SurfaceRecombination"   Parameter = "S0_e"
   Region = "bulk"  Model = LatticeHeatCapacity    Parameter = cv
   RegionInterface = "Region.0/Region.1"    Model = "SurfaceRecombination"   Parameter = "S0_h"
}
```

2.8 NonLocalPlot section

The `NonLocalPlot` section is used to visualize data defined on nonlocal lines (see Section 2.10.7.3 on page 15.85).

2.9 Solve section

The `Solve` section is the only section in which the order of commands and their hierarchy are important. It consists of a series of simulation commands to be performed that are activated sequentially, according to the order of commands in the input file. Many `Solve` commands are high-level commands that have lower level commands as parameters.

Figure 15.17 on page 15.55 shows an example of these different command levels:

- The `Coupled` command (the base command) is used to solve a set of equations.
- The `Plugin` command is used to iterate between a number of coupled equations.

- The `Quasistationary` command is used to ramp a solution from one boundary condition to another.
- The `Transient` command is used to run a transient simulation.

Furthermore, small-signal AC analysis can be performed with the `ACCoupled` command. An advanced ramping by continuation method can be performed with the command `Continuation`. (The `ACCoupled` and `Continuation` commands are presented in Section 3.6 on page 15.113.)

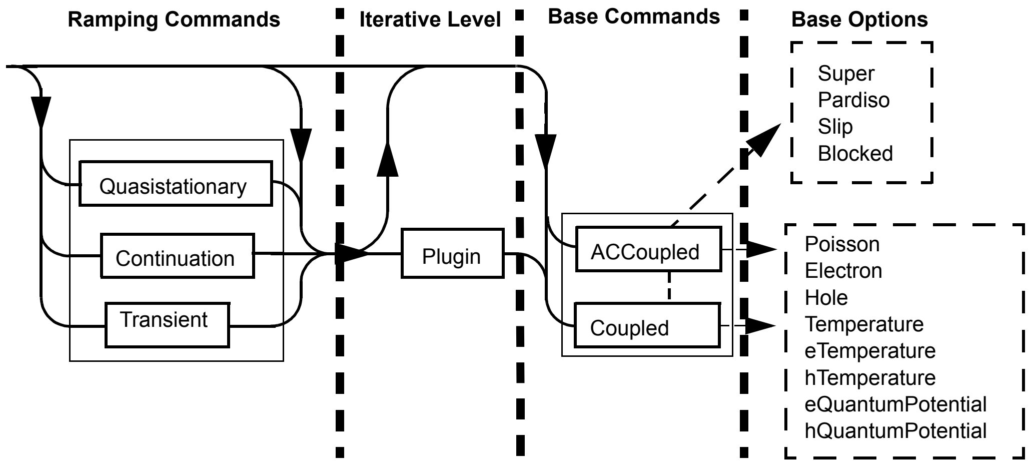


Figure 15.17 Different levels of Solve commands

2.9.1 Coupled command

The `Coupled` command activates a Newton-like solver over a set of equation–variable pairs. In DESSIS, the basic semiconductor model equations are the Poisson equation, the two continuity equations, and the different thermal and energy equations. Table 15.13 presents a list of the equation–variable pairs that can be used in a `Coupled` command (see Table 15.48 on page 15.114).

Table 15.13 Equation–variable pairs for Coupled command

Keyword	Corresponding equation	Corresponding variable
`Electron`	Electron continuity	Electron density
`Hole`	Hole continuity	Hole density
`Poisson`	Poisson	Electrostatic potential
`Temperature`	Temperature	Temperature
`eTemperature`	Electron temperature	Electron temperature
`hTemperature`	Hole temperature	Hole temperature
`eQuantumPotential`	Electron quantum potential	Electron quantum potential
`hQuantumPotential`	Hole quantum potential	Hole quantum potential

The syntax of the `Coupled` command is:

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

or equivalently:

```
<equation-variable>
```

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

```
Coupled { Poisson Electron }
```

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

If the following command is used:

```
Poisson
```

only the electrostatic potential is computed using the Poisson equation.

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

- The maximum number of iterations allowed.
- The desired precision of the solution.
- The linear solver that must be used.
- Whether the solution is allowed to worsen over a number of iterations.

These parameters are summarized in Table 15.14. The command is controlled by both an absolute criterion and a relative error criterion. The relative error control can be specified with the optional parameter `Digits`. The absolute error control can be specified in the `Math` section (see Section 2.10 on page 15.73).

Table 15.14 Optional parameters for Coupled command

Parameter	Description
`Iterations = <integer>`	Maximum number of iterations of the Newton algorithm.
`Digits = <float>`	Loops until this relative precision is reached.
`Method = <linear solver>`	Specifies the linear solver to be used.
`SubMethod = <linear solver>`	For a two-level `Blocked` linear solver, this specifies the second linear solver to use for the inner loop.
`NotDamped = <integer>`	Allows a solution to worsen for a specified number of iterations. If specified, after the given number of undamped iterations, the Newton steps are limited by the Bank–Rose algorithm so that the solution error does not worsen, that is, the solution is damped.
`LineSearchDamping = <float>`	Sets the minimal damping coefficient for line search damping. `<float>` must be greater than zero and not greater than 1. A value of 1 disables line search damping.

The following example limits the previous `Coupled { Poisson Electron }` example to ten iterations and uses the Slip linear solver:

```
Coupled ( Iterations=10 Method=Slip ) { Poisson Electron }
```

NOTE Use a large number of iterations when the coupled iteration is not inside a ramping process. In this context, allow the Newton algorithm to proceed as far as possible. Inside a ramping command (for example, `Quasistat`, `Transient`), the maximum number of iterations must be limited to approximately ten because if the Newton process does not converge rapidly, it is preferable to try again with a smaller step size than to pursue an iterative process that is not likely to converge.

The linear method used in a coupled iteration depends on the type and size of the problem solved. Table 15.15 lists the basic rules for making this decision. Line search damping (option `LineSearchDamping`) and Bank–Rose damping (option `NotDamped`) are unrelated. Unlike the right-hand side that DESSIS shows in the solve report, line search damping is based on the Fedorenko norm. Therefore, despite damping, the right-hand side in the solve report can increase.

Table 15.15 Hints on choice of linear solver

Solver type	Problem type
`Blocked`	Multiple device problems (mixed-mode simulations).
`ILS`	Single or multiple device problems with over 8000 vertices.
`ParDiSo`	Single device problems with meshes up to 10000 vertices.
`Slip`	Single device problems with over 8000 vertices.
`Super`	Single device problems with meshes up to 10000 vertices.
`UMF`	Single or multiple device problems with up to 10000 total mesh vertices.

2.9.2 Plugin command

The `Plugin` command controls an iterative loop over two or more `Coupled` commands. It is used when a fully coupled method would use too many resources of a given machine, or when the problem is not yet solved and a full coupling of the equations would diverge. The `Plugin` syntax is defined as:

```
Plugin ( <optional parameters> ) ( <list-of-coupled-commands> )
```

`Plugin` commands can have any complexity but, usually, only a few combinations are effective. One standard form is the Gummel iteration in which each basic semiconductor equation is solved consecutively. With the `Plugin` command, this is written as:

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

or using the abbreviated `Coupled` command, as:

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

As the `Plugin` command loops through a number of `Coupled` commands, it takes as its parameters:

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

Table 15.16 lists the corresponding keywords of these parameters.

Table 15.16 Parameters for Plugin command

Parameter	Description
`BreakOnFailure`	Instructs DESSIS to stop when there is a bad convergence report if an inner `Coupled` fails.
`Digits = <float>`	Specifies the number of digits of precision for the solution.
`Iterations = <integer>`	Sets the maximum number of iterations over the `Plugin` loop. The special value of 0 is used to perform one loop without error testing.

NOTE `Plugin` commands can be used with other `Plugin` commands, such as: `Plugin{ Plugin{ ... } Plugin { ... } }`. Figure 15.18 illustrates the corresponding loop structure. A hierarchy of `Plugin` commands allows more complex iterative solve patterns to be created.

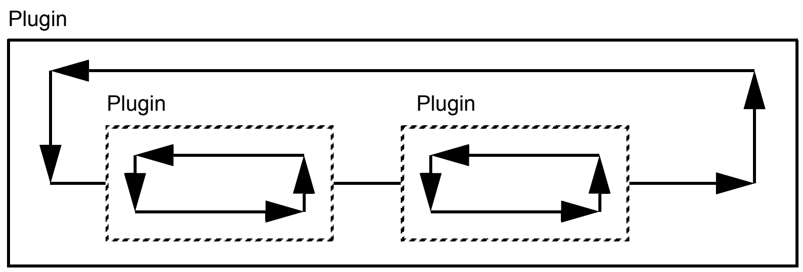


Figure 15.18 Example of hierarchy of Plugin commands

2.9.3 Quasistationary command

The `Quasistationary` command is used to ramp a device from one solution to another through the modification of its boundary conditions (for example, ramping the voltage at a contact) or parameter values. The command must start with a device that has been solved already.

The simulation continues by iterating between the modification of the boundary conditions or parameter values, and re-solving the device (see Figure 15.19). The command to re-solve the device at each iteration is given with the `Quasistationary` command.

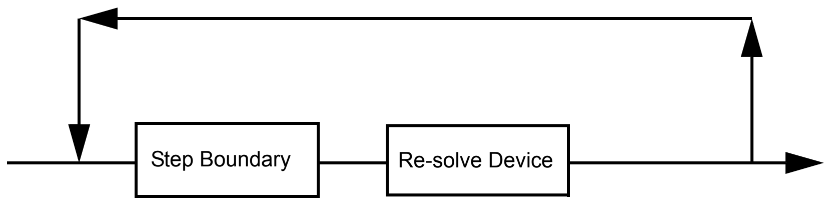


Figure 15.19 Structure of Quasistationary command

2.9.3.1 Ramping boundary conditions

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

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

The possible parameters are listed in Table 15.17, and the solve command is `Coupled`, `Plugin`, or possibly another `Quasistationary`.

Table 15.17 Base parameters for Quasistationary command

Parameter	Description
`Goal {Voltage = <float> name = <string>},` `Goal {Current = <float> name = <string>},` `Goal {Temperature = <float> name = <string>},` `Goal {Power = <float> name = <string>}`	Sets the new target values for a specified contact (`name`). Contact type can be a fixed voltage (`Voltage`), current source (`Current`), fixed temperature (`Temperature`), or heat source (`Power`).
`InitialStep = <float>`	Initial step size (default is 0.1).
`MaxStep = <float>`	Maximum step size (default is 1.0).
`MinStep = <float>`	Minimum step size (default is 1.0×10^{-3}).
`Increment = <float>`	The step size is multiplied by this value before being added to the current `t` value if the equation(s) for the current `t` is solved successfully (default is 2).
`Decrement = <float>`	The step size is divided by this value before being added to the previously successful value `t` if the equation(s) for the current `t` is not solved successfully (default is 2).
`DoZero`	The equation section is solved for `t=0`.

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

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

Internally, the `Quasistationary` command works by ramping a variable `t` from 0.0 to 1.0. This means that the corresponding voltage at the contact changes according to the formula **V= V0 + t (V1 – V0)** where **V0** is the initial voltage and **V1** is the final voltage, which is specified in the `Goal` statement.

Greater control of the command is possible with the `Step` control parameters that affect the behavior of the `t` variable. (The control is not made over contact values because more than one contact can be ramped simultaneously.)

Step control parameters are `InitialStep`, `MaxStep`, `MinStep`, `Increment`, and `Decrement`. `InitialStep` controls the size of the first step of the ramping. The step size is automatically augmented or reduced depending on the rate of success of the inner solve command. `MaxStep` and `MinStep` limit this change. The rate of increase is controlled by the number of performed Newton iterations, and by the factor `Increment`.The step size is only reduced by the factor `Decrement`, when the inner solve fails. Consequently, the ramping process stops when the `MinStep` condition is reached.

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

Each contact has a type, which can be voltage, current, or charge. Each `Quasistationary` command has a goal, which can also be voltage, current, or charge. The goal and contact type must match. If they do not match, DESSIS changes the contact type to match the goal. If the goal is current and the contact type is voltage, DESSIS changes the contact type to current. If the goal is voltage and the contact type is current, DESSIS changes the contact type to voltage. However, DESSIS cannot change a contact of charge type to another type.

The initial value (for t=0) is the current or voltage computed for the contact before the `Quasistationary` command starts. Contacts keep their boundary condition type after the `Quasistationary` command finishes. To change the boundary condition type of a contact explicitly, use the `Set` command (see Section 2.9.10 on page 15.73).

2.9.3.2 Ramping physical parameter values

A `Quasistationary` command allows parameters from the DESSIS parameter file to be ramped. The `Goal` statement has the form:

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

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

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

```
dessis --parameter-names
```

This list of parameters corresponds to those in the DESSIS parameter file, which can be obtained by using `dessis -P` (see Section 2.13.2 on page 15.91).

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

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

DESSIS reads the devices in the command file and reports all parameter names that can be ramped. However, no simulation is performed. The models in Table 15.18 contain command file parameters that can be ramped.

Table 15.18 Command file parameters

Model name	**Parameters**
`OptBeam(<index>)` `RayBeam(<index>)`	`RefractiveIndex, SemAbs, SemSurf, SemVelocity_Vx, SemVelocity_Vy, SemVelocity_Vz, SemWindow_xmax, SemWindow_xmin, SemWindow_ymax, SemWindow_ymin, SemWindow_zmax, SemWindow_zmin, WaveDir_x, WaveDir_y, WaveDir_z, WaveEnergy, WaveInt, WaveLength, WavePower, WaveTime_tmax, WaveTime_tmin, WaveTsigma, WaveXYsigma`
`RadiationBeam`	`Dose, DoseRate, DoseTSigma, DoseTime_end, DoseTime_start`
`Traps(<index>)`	`Conc, EnergyMid, EnergySig, eGfactor, eJfactor, eXsection, hGfactor, hJfactor, hXsection`
`DeviceTemperature`	`Temperature`

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

```
Model="OptBeam(0)"
Model="Traps(1)"
```

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

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

Mole fraction–dependent parameters can be ramped. For example, if the parameter `p` is mole fraction–dependent, the parameter names listed in Table 15.19 can appear in a `Goal` statement.

Table 15.19 Mole fraction–dependent parameters as Quasistationary Goals

Parameter in Goal statement	Description
`Parameter=p`	Parameter `p` in non-mole fraction–dependent materials.

`Parameter="p(0)"`
`Parameter="p(1)"`
`...`
	Interpolation value of `p` at Xmax(0), Xmax(1), …

`Parameter="B(p(1))"`
`Parameter="C(p(1))"`
`Parameter="B(p(2))"`
`Parameter="C(p(2))"`
`...`
	Quadratic and cubic interpolation coefficients of `p` in intervals [Xmax(0), Xmax(1)], [Xmax(1), Xmax(2)], …

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

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

Parameters in PMI models can also be ramped.

2.9.3.3 Saving and plotting data during a Quasistationary solve sequence

Data can be saved and plotted during a `Quasistationary` ramping process by using the `Plot` command. Table 15.20 shows how the `Plot` command is specified.

Table 15.20 Plot parameter in Quasistationary command

`Plot { Range = (<float> <float>)`
`Intervals = <integer> }`
	Specifies at which internal variable `t` (between 0 and 1) the save and plot files are saved.

The `Plot` parameter is placed with the other `Quasistationary` parameters, for example:

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

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

Another way to plot data is with the `Plot` statement in the `Solve` command rather than inside the `Quasistationary` statement (see Section 2.9.6 on page 15.66). This command is added after the given `Solve` command, such as:

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

2.9.4 Transient command

The `Transient` command is used to perform a transient time simulation. The command must start with a device that has already been solved. The simulation continues by iterating between incrementing time and re-solving the device (see Figure 15.20). The command to solve the device at each iteration is given with the `Transient` command.

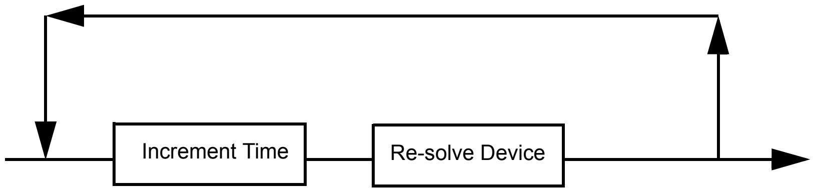


Figure 15.20 Structure of Transient command

The syntax of the `Transient` command is:

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

Table 15.21 lists the possible parameters. The solve command is `Coupled` or `Plugin`.

Table 15.21 Base parameters for Transient command

Parameter	Description
`InitialTime = <float>`	Start time of the simulation (default is 0.0 s).
`FinalTime = <float>`	Final time of the simulation.
`InitialStep = <float>`	Initial step size (default is 0.1 s).
`MaxStep = <float>`	Maximum step size (default is 1.0 s).
`MinStep = <float>`	Minimum step size (default is 1.0×10^{-3} s).

An example of performing a transient simulation for $10\ \mu s$ is:

```
Transient( InitialTime = 0.0 FinalTime=1.0e-5 ){
   Coupled { Poisson Electron Hole }
}
```

The `Transient` command allows the user to overwrite time-step control parameters, which have default values or are globally defined in the `Math` section. The error control parameters that are accepted by the `Transient` command are listed in Table 15.22. The parameters `TransientError`, `TransientErrRef`, and `TransientDigits` control the error over the transient integration method.

NOTE This differs from the error control for the `Coupled` command, which only controls the error of each nonlinear solution. As with the error parameters for the `Coupled` command, the transient error controls can be both absolute and relative. Absolute values are parameterized according to equation–variable type.

Table 15.22 Error control parameters for Transient command

Parameter	Description
`TransientDigits = <float>`	Overwrites the number of relative digits required for the transient control.
`TransientErrRef(<equation-pair>)=<float>`	Overwrites the error reference in time-step control for a given equation pair (for example, `Poisson`, `Electron`).
`TransientError(<equation-pair>)=<float>`	Overwrites the absolute transient error in time-step control for a given equation pair (for example, `Poisson`, `Electron`).
`TrStepRejectionFactor`	Overwrites the value of f_{rej} factor (see Section 32.4.3 on page 15.529).
`CheckTransientError | NoCheckTransientError`	Enables/disables all error controls through the transient integration method. If disabled, error control is managed only by the convergence property of the inner solve method.

The plot controls of the `Transient` command and `Quasistationary` command are identical, except that the parameter `t` is identical to the time.

In two similar examples, the `Plot` parameter is placed with the other `Transient` parameters, for example:

```
Transient( InitialTime = 0.0 FinalTime = 1.0e-5
    Plot { Range = (0 3.0e-6) Intervals=3 } )
    { Coupled { Poisson Electron Hole } }
```

This example saves four plot files at `t = 0.0`, `1.0e-6`, `2.0e-6`, and `3.0e-6`.

By placing `Plot` in the `Solve` section (see Section 2.9.6 on page 15.66), this example can be rewritten:

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

2.9.5 Large signal cyclic analysis

For high-speed and high-frequency operations, devices are often evaluated by cyclic biases. After a time, device variables change periodically. This cyclic-bias steady state [129] is a condition that occurs when all parameters of a simulated system return to the initial values after one cycle bias is applied.

In fact, such a cyclic steady state is reached by using standard transient simulation. However, this is not always effective, especially if some processes in the system have very long characteristic times in comparison with the period of the signal.

For example, deep traps usually have relatively long characteristic times. A suggested approach [130] allows for significant acceleration of the process of reaching a cyclic steady state solution. The approach is based on iterative correction of the initial guess at the beginning of each period of transient simulation, using previous initial guesses and focusing on reaching a cyclic steady state. This approach is implemented in DESSIS.

2.9.5.1 Description of the method

The original method [130] is summarized. Transient simulation starts from some initial guess. A few periods of transient simulation are performed and, after each period, the change over the period of each independent variable of the simulated system is estimated (that is, the potential at each vertex of all devices, electron and hole concentrations, carrier temperatures, and lattice temperature if hydrodynamic or thermodynamic models are selected, trap occupation probabilities for each trap type and occupation level, and circuit node potentials in the case of mixed-mode simulation).

If x_n^I denotes the value of any variable in the beginning of the n-th period, and x_n^F denotes the same value at the end of the period, the cyclic steady state is reached when:

$$\Delta x_n = x^F_n - x^I_n \tag{15.2}$$

is equal to zero. Considering linear extrapolation and that the goal is to achieve $\Delta x_{n+1} = 0$, the next initial guess can be estimated as:

$$x^I_{n+1} = x^I_n - \gamma \frac{x^I_n - x^I_{n-1}}{\Delta x_n - \Delta x_{n-1}} \Delta x_n \tag{15.3}$$

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

As (Eq. 15.3) contains uncertainty such as 0/0, especially when Δx is close to zero (when the solution is close to the steady state), special precautions are necessary to provide robustness of the algorithm.

Consider the derivation of (Eq. 15.3) in a different fashion – near the cyclic steady state. If such a steady state exists, the initial guess $y = x^I$ is expected to behave with time as $y = a\exp(-\alpha t) + b$, where $\alpha > 0$. It is easy to show that (Eq. 15.3) gives $x^I = b$, that is, a desirable cyclic steady state solution. It follows that the ratio r:

$$r = -\frac{x^I_n - x^I_{n-1}}{\Delta x_n - \Delta x_{n-1}} \tag{15.4}$$

can be estimated as $r \approx 1/(1 - \exp(-\alpha t))$. From this, it is clear that because α is positive, the condition $r \geq 1$ must be valid. Moreover, r can be very large if some internal characteristic time (like the trap characteristic time) is much longer than the period of the cycle. Using the definition of r from (Eq. 15.4), (Eq. 15.3) can be rewritten as:

$$x^I_{n+1} = x^I_n + \gamma r \Delta x_n \tag{15.5}$$

Although (Eq. 15.4) and (Eq. 15.5) are equivalent to (Eq. 15.3), it is more convenient inside DESSIS to use (Eq. 15.4) and (Eq. 15.5). DESSIS never allows r to be less than 1 because of the above arguments.

To provide convergence and robustness, it is reasonable also not to allow r to become very large. In DESSIS, r cannot exceed a user-specified parameter $r_{\max}$. The user can also specify the value of the parameter $r_{\min}$.

An extrapolation procedure, which is described by (Eq. 15.4) and (Eq. 15.5), is performed for every variable of all the devices, in each vertex of the mesh. Instead of densities, which can spatially vary over the device by many orders of magnitude, the extrapolation procedure is applied to the appropriate quasi-Fermi potentials.

For the trap equations, extrapolation can be applied either to the trap occupation probability f_T (the default) or, optionally, to the 'trap quasi-Fermi level,' $\psi_T = -\ln((1-f_T)/f_T)$. The cyclic steady state is supposedly reached if the following condition is satisfied:

$$\frac{\Delta x}{x + x_{ref}} < \varepsilon_{cyc} \tag{15.6}$$

Values of x_{ref} are the same as `ErrRef` values of the `Math` section. For every object of the simulated system (that is, every variable of all devices), an averaged value r_{av}^{ob} of the ratio r is estimated and can be optionally printed. Estimation of r_{av}^{ob} is performed only at such vertices of the object, where the condition:

$$\frac{\Delta x}{x + x_{ref}} < \frac{\varepsilon_{cyc}}{f} \tag{15.7}$$

is fulfilled, that is, the same condition as (Eq. 15.6), but with a possibly different tolerance $\varepsilon 1_{cyc} = \varepsilon_{cyc}/f$.

The following extrapolation procedures are allowed:

1. Use of averaged extrapolation factors for every object. This is the default option.
2. Use of the factor r independently for every mesh vertex of all objects. If for some reason, the criterion in (Eq. 15.7) is already reached, the value of factor r is replaced by the user-defined parameter $r_{\min}$. The option is activated by the keyword `-Average` in the `Extrapolate` statement inside `Cyclic` specification.
3. The same as Step 2, but for the points where (Eq. 15.7) is fulfilled, the value of factor r is replaced by the averaged factor r_{av}^{ob}. The option is activated by the keyword `-Average` in the `Extrapolate` statement inside the `Cyclic` specification, accompanied by specification of the parameter $r_{\min} = 0$.

2.9.5.2 Syntax and implementation

Cyclic analysis is activated by specifying the parameter `Cyclic` in the parameter list of the `Transient` statement. `Cyclic` is a complex structure–like parameter and contains cyclic options and parameters in parentheses:

```
Transient( InitialTime=0 FinalTime=2.e-7 InitialStep=2.e-14 MinStep=1.e-16
           Cyclic( <cyclic-parameters> )
) { ... }
```

Table 15.23 lists the available `Cyclic` parameters.

Table 15.23 Parameters for cyclic analysis

Parameter	Description
`Period = <float>`	Defines the period of the cycle [s]. The default is no cyclic analysis.
`StartingPeriod = <integer>`	Defines the number of the period from which the extrapolation procedure starts. It cannot be less than 2 (the default).

Table 15.23 Parameters for cyclic analysis

Parameter	Description
`Accuracy = <float>`	Defines the tolerance ε_{cyc} .
`RelFactor = <float>`	Defines the relaxation factor Υ .
`Extrapolate()`	Defines, in parentheses, optional parameters for cyclic extrapolation.

In the parentheses of the optional parameter `Extrapolate` (in a `Cyclic` statement), additional options of the cyclic extrapolation procedure are defined. Table 15.24 lists these options.

Table 15.24 Optional parameters of Extrapolate statement in cyclic specification

Parameter	Description
`Forward <bool>`	DESSIS proceeds as in a standard transient, that is, without cyclic extrapolation procedure. The default is false.
`QFtraps <bool>`	For trap probabilities, the extrapolation procedure applies to 'trap quasi-Fermi level' ψ_T instead of trap occupation probabilities f_T . The default is false.

`Average <bool>`
`-Average`
	For each object, averaged factor r_{av}^{ob} is used. The default is true.
	Factor r is defined separately for each vertex.

`Factor = <float>`	Defines the coefficient f for r_{av}^{ob} estimation. The default is 1.
`MaxVal = <float>`	Defines r_{max}. The default is 25.
`MinVal = <float>`	Defines r_{min}. The default is 1.
`Print <bool>`	Averaged factors r_{av}^{ob} for each object are printed.

An example of a `Transient` command with `Cyclic` specification is:

```
Transient( InitialTime=0 FinalTime=2.e-7 InitialStep=2.e-14 MinStep=1.e-16
          Cyclic( Period=8.e-10 StartingPeriod=4
                 Accuracy=1.e-4 RelFactor=1
                 Extrapolate (Average Print MaxVal=50) )
) { ... }
```

NOTE The value of the parameter `Period` in the `Cyclic` statement must be divisible by the period of the bias signal. A periodic bias signal must be specified elsewhere in the input file.

2.9.6 Plot, Save, and Load commands

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

The `Load` statement can only be used as a base level `Solve` statement. For example, in the case of a transient simulation, `Load` can only be used before or after the transient, not during it.

The `Save` statement generates files of type `.sav`. The `Plot` statement generates files of type `.dat`. An important distinction between these file types is that a `.sav` file automatically contains all contact biases and currents. The biases are reloaded in the `Load` statement, which means:

- Contact biases specified in the `Electrode` section are overwritten after loading the `.sav` file. This statement does not apply if a `.dat` file is loaded.
- Multiple load and save operations are allowed in one input file.

The commands are defined as:

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

where:

`<command>` is either `Plot`, `Save`, or `Load`. `Plot` refers to data specified in the `Plot` section.

`<parameters-opt>` is a list of the options delimited by parentheses (see Table 15.25).

Table 15.25 Optional parameters for Plot, Save, and Load commands

Parameter	Description
`FilePrefix = "<fileprefix>"`	The prefix of the file name to which data is saved or from which graphs are plotted. The file names consist of the file prefix, the instance name, an optional local number (depending on the option `Overwrite/noOverwrite`), and the extension `_des.sav`. The default file prefix is `save<globalsaveindex>` for `Save` and `plot<globalplotindex>` for `Plot`.
`Time = (list of time entries)`	A list of the times when data is saved or plotted. Time entries are separated by semicolons (see Section 2.9.6.1 on page 15.69).
`Iterations = (list of numbers)`	Used in plugins to save or plot at certain iterations. Numbers are separated by semicolons. The default is to save or plot for all iterations.
`IterationStep = <number>`	Used in plugins, quasistationaries, continuations, and transients to save or plot all `<number>` steps.
`Compressed | unCompressed`	Specifies if save or plot files are written in a compressed or uncompressed format (with the extensions `_des.sav.Z` or `_des.dat.Z`, respectively). The default is `unCompressed`. This option must not be used if DESSIS runs as an application of GENESISe.
`Number = <number>`	When a sequence of solutions is saved during a `Quasistationary` or `Transient` simulation, DESSIS automatically indexes the file names (for example, `n3_0001_des.sav`, `n3_0002_des.sav`, ...). The keyword `Number` specification is essential in a `Load` command to reload a specific solution (in the example, `Number=2` reloads the solution file `n3_0002_des.sav`).
`Voltage ( Intervals = <number> )`	Only for `Continuation` simulations. The specified voltage range (that is, `MinVoltage` and `MaxVoltage` of `Continuation`) is divided into intervals, and the save or plot files are written every time the voltage enters one of these new intervals.
`Current ( Intervals = <number> )`	Only for `Continuation` simulations. Similar to the `voltage` specification shown previously. If `LogCurrent` is given in a `Continuation` simulation, a log current range is used.

Table 15.25 Optional parameters for Plot, Save, and Load commands

Parameter	Description
`Voltage ( Difference = <value> )`	Only for `Continuation` simulations. Similar to the `voltage` specification shown previously. In this example, `Difference` specifies the intervals in terms of positive voltage differences [V].
`Current ( Difference = <value> | LogDifference = <value> )`	Only for `Continuation` simulations. Similar to the `voltage` specification shown previously. `Difference` [A]; `LogDifference` is the logarithm of the current difference [A]. The value must be positive. The specification of `LogDifference` or `Difference` is independent of the keyword `LogCurrent` in the `Continuation` parameters.
`Overwrite/noOverwrite`	`Overwrite` allows each save or plot file to rewrite over the same file name at each loop. `NoOverwrite` forces each new save or plot file name to be given a new name by numbering the files. The default is `Overwrite`.

`When (Contact = <contact name> Voltage = <voltage>)`
`When (Contact = <instance name>.<contact name> Voltage = <voltage>)`
`When (Contact = <contact name> Current = <current>)`
`When (Contact = <instance name>.<contact name> Current = <current>)`
`When (Node = <node name> Voltage = <voltage>)`
	These options are available for `Plot`, `Save`, and `CurrentPlot` commands inside a `Quasistationary`, `Transient`, or `Continuation` statement. They are used to monitor the (inner) voltage at a contact, the current at a contact, or the voltage at a circuit node, and to plot or save whenever this value exceeds a threshold.
	If the instance name is omitted, DESSIS uses the empty name (" "). This is a shortcut for pure device simulations (no `System` section).
	A plot or save occurs whenever one of these conditions applies:
	`vc(last iteration)<threshold≤vc(this iteration)`
	`vc(last iteration)>threshold≥vc(this iteration)`
	Here, `vc(iteration)` denotes the voltage or current for a given iteration in the enclosing statement. Consequently, plots or saves may occur both for increasing and decreasing values.

NOTE The plot parameters in Table 15.25 are also available in the `ACCompute` option discussed in Section 3.8.3 on page 15.117.

Use `<system-opt>` to obtain a list of optional entries delimited by braces. Table 15.26 lists these optional entries.

Table 15.26 System options for Plot, Save, and Load commands

Option	Description
`<device-name>`	Selected if the device `<device-name>` is to be plotted, saved, or loaded.
`Circuit`	Selected if the `circuit` devices and circuitry are to be saved or loaded (not for plot) (see Chapter 3 on page 15.101).

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

2.9.6.1 Example: Solve section of input file with Plot, Save, and Load operations

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

The first `Plot` statement in `Plugin` writes (after the computation of the Poisson equation) to a file named `output/poisson_des.dat`. The second `Plot` statement in `Plugin` writes to a file called `output/electric_0000_des.dat` and increases the internal number for each call.

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

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

The `Plot` statement in this transient simulation example specifies three different types of time entries (separated by semicolons). The first entry indicates all the times within this range when a plot file must be written. The second time entry forces the transient simulation to compute solutions for the given times. In this example, the given times are 0.25, 0.5, 0.75, and 1.0. The third entry is for the single time of 0.7. This format is similar to quasistationaries.

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

2.9.6.2 Example: Solve section of input file with multiple Save and Load operations

```
...
Solve {
      ...
    # Ramp the gate and save structures
      # First gate voltage
      Quasistationary (InitialStep=0.1 MaxStep=0.1 MinStep=0.01
                          Goal {Name="gate" Voltage=1})
                          {Coupled {Poisson Electron Hole}}
                       Save(FilePrefix="vg1")
      # Second gate voltage
```

```
        Quasistationary (InitialStep=0.1 Maxstep=0.1 MinStep=0.01
                         Goal {Name="gate" Voltage=3})
                         {Coupled {Poisson Electron Hole}}
                         Save(FilePrefix="vg2")
     # Load the saved structures and ramp the drain
        # First curve
        Load(FilePrefix="vg1")
        NewCurrentPrefix="Curve1"
        Quasistationary (InitialStep=0.1 MaxStep=0.5 MinStep=0.01
                         Goal {Name="drain" Voltage=2.0})
                         {Coupled {Poisson Electron Hole}}
        # Second curve
        Load(FilePrefix="vg2")
        NewCurrentPrefix="Curve2"
        Quasistationary (InitialStep=0.1 MaxStep=0.5 MinStep=0.01
                         Goal {Name="drain" Voltage=2.0})
                         {Coupled {Poisson Electron Hole}}
}
...
```

2.9.7 System command

The `System` command allows UNIX commands to be executed during a DESSIS simulation:

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

The `System` command can appear as an independent command in the `Solve` section, as well as within a `Transient`, `Continuation`, `Plugin`, or `Quasistationary` command. The string argument of the `System` command is passed to a UNIX shell for evaluation.

By default, the return status of the UNIX command is ignored. If the variant:

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

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

2.9.8 NewCurrentPrefix statement

By default, DESSIS saves all current plots in one file (as defined by the variable `Current` in the `File` section). This behavior can be modified by the `NewCurrentPrefix` statement in the `Solve` section:

```
NewCurrentPrefix = prefix
```

This statement appends the given prefix to the default, current file name, and all subsequent `Plot` statements are directed to the new file. Multiple `NewCurrentPrefix` statements can appear in the `Solve` section, for example:

```
Solve {
   Circuit
   Poisson
   NewCurrentPrefix = "pre1"
   Coupled {Poisson Electron Hole Contact Circuit}
   NewCurrentPrefix = "pre2"
   Transient (
      MaxStep= 2.5e-6 InitialStep=1.0e-6
      InitialTime=0.0 FinalTime=0.0001
```

```
            Plot {range=(10e-6,40e-6) Intervals=10}
        )
        {Coupled {Poisson Electron Hole Contact Circuit}}
    }
```

In this example, the current files specified in the `File` and `System` sections contain the results of the `Circuit` and `Poisson` solves. The results of the `Coupled` solution are saved in a new current file with the same name but prefixed with `pre1`. The last current file contains the results of the `Transient` solve with the prefix `pre2`.

NOTE The file names for current plots defined in the `System` section, and the plot files of AC analyses are also modified by a `NewCurrentPrefix` statement.

2.9.9 CurrentPlot section

The `CurrentPlot` statement in the `Solve` section provides full control over the plotting of device currents and circuit currents. By default, currents are plotted after each iteration in a `Plugin`, `Quasistationary`, or `Transient` command. This behavior can be modified by a `CurrentPlot` statement in the body of these commands. If a `CurrentPlot` statement is present, it determines the exact location of all plot points that are written to the `.plt` file. DESSIS can still perform computations for some intermediate points, but they are not written to the file.

NOTE Do not confuse the `CurrentPlot` statement in the `Solve` section with the `CurrentPlot` section as described in Section 2.7 on page 15.52.

The syntax of the `CurrentPlot` statement is:

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

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

`Time = (<float> ; <float> ; <float> ;)`
The list of time entries enumerates the times for which a current plot is requested. The entries are separated by semicolons. A time entry can have these forms:

`floating point number` The time value for which a current plot is requested.

`Range = (a b)` This option specifies a free plot range between `a` and `b`. All the time points in this range are plotted.

`Range = (a b) Intervals = n`
This option specifies `n` intervals in the range between `a` and `b`. In other words, these plot points are generated:

$$t = a, t = a + \frac{b-a}{n}, \ldots, t = b - \frac{b-a}{n}, t = b \qquad (15.8)$$

`Iterations = (<integer>; <integer>; <integer>;)`
The list of integers specifies the iterations for which a plot is required. This option is available for the `Plugin` command.

`IterationStep = <integer>`

This option requests a current plot every `n` iterations. It is available for the `Plugin` command.

`When (<when condition>)`

A `When` option can be used to request a current plot whenever a condition has been met. This option works in the same way as in a `Plot` or `Save` command (see Table 15.25 on page 15.67).

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

2.9.9.1 Example: CurrentPlot statements

A `CurrentPlot` statement by itself creates a current plot for each iteration:

```
Quasistationary (InitialStep=0.2 MinStep=0.2 MaxStep=0.2
Goal { Name="drain" Voltage=0.5 })
   { Coupled { Poisson Electron Hole }
   CurrentPlot }
```

If no current plots are desired, a current plot for the ‘impossible’ time $t = -1$ can be specified:

```
Quasistationary (InitialStep=0.2 MinStep=0.2 MaxStep=0.2
Goal { Name ="drain" Voltage=0.5 })
   { Coupled { Poisson Electron Hole }
   CurrentPlot ( Time = (-1)) }
```

In this example, the currents of the device `nmos` are plotted for t=0, $t=10^{-8}$, and $t=10^{-7}$:

```
Transient ( MaxStep=1e-8 InitialTime=0 FinalTime=1e-6 )
         { Coupled { Poisson Circuit }
         CurrentPlot ( Time = (0 ; 1e-8; 1e-7)) { nmos } }
```

This `CurrentPlot` statement produces 11 equidistant plot points in the interval 0, 10^{-5} :

```
Transient ( MaxStep = 1e-8 InitialTime=0 FinalTime=1e-5 )
         { Coupled { Poisson Circuit }
         CurrentPlot ( Time = (range = (0 1e-5) intervals = 10)) }
```

In this example, a current plot for iteration 1, 2, 3, and for every tenth iteration is specified:

```
Plugin { Poisson Electron Hole
      CurrentPlot ( Iterations = (1; 2; 3) IterationStep = 10 ) }
```

A `CurrentPlot` statement can also appear at the top level in the `Solve` section. In this case, the currents are plotted when the flow of control reaches the statement.

A `CurrentPlot` statement is also recognized within a `Continuation` command. In this case, the time in the `CurrentPlot` statement corresponds to the arc length in the `Continuation` command.

However, only free plot ranges are supported.

2.9.10 Set command

The `Set` command changes the boundary condition type for an electrode. The `Electrode` section (see Section 2.3 on page 15.39) specifies the initial boundary condition. To change the boundary condition of a current contact `<name>` to voltage type, use `Set(<name> mode voltage)`. To change the boundary condition of a voltage contact `<name>` to current type, use `Set(<name> mode current)`. For example, use:

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

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

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

An alternative method to change the boundary condition type of a contact is to use the `Quasistationary` statement (see Section 2.9.3 on page 15.58). This alternative is usually more convenient. However, a goal value for the boundary condition must be specified, whereas the `Set` command allows the user to fix the current or voltage at a contact to a value reached during the simulation, even if this value is not known beforehand. In mixed-mode simulations, the `Set` command can be used to determine the boundary conditions at nodes (see Section 3.4.4 on page 15.110 and Section 3.8.6 on page 15.121).

2.10 Math section

The `Math` section is used to specify defaults for the different `Solve` commands. The two types of `Math` entries are device-specific and global. Device-specific entries refer to parameters that affect the solve methods of a device. Global entries are device independent and affect the global solution methods.

2.10.1 Device-specific Math keywords

Device-specific keywords are used in the `Math` section and device-specific `Math` sections. In this section, keywords are grouped by functionality. The `Math` parameters for the physics can be placed in the `Physics` section; however, the parameters are in the `Math` section because they strongly affect the math. Table 15.27 on page 15.74 lists four numeric performance keywords:

- `Cylindrical` specifies that the device is simulated using cylindrical coordinates. In this case, a 3D device is specified by a 2D mesh and the vertical axis around which the device is rotated.
- `EdgeMagneticDiscretization` specifies that the device is to respond to external magnetic fields.
- `RecomputeQFP` recomputes the quasi-Fermi potentials when the electrostatic potential changes.
- `ComputeIonizationIntegrals` computes the ionization integrals from the local field maxima.

Table 15.27 Math parameters for physics

Parameter	Description
`ComputeIonizationIntegrals (flags)`	Switches on the computation of ionization integrals for paths that cross local field maxima in a semiconductor.
`Cylindrical (<float>)`	Cylindrical coordinates are used to simulate a 2D device. This allows a 2D device to be rotated around a vertical axis. The optional argument is the horizontal coordinate of the axis of rotation (default is 0), which must be less than or equal to the smallest horizontal device coordinate. It is switched off by default.
`EdgeMagneticDiscretization`	Switches on the discretization of the carrier current densities used for galvanometric transport equations. This keyword must be specified for nonzero magnetic simulations.
`RecomputeQFP`	Keeps density variables constant, and recomputes quasi-Fermi potentials when the electrostatic potential changes and carrier equations are not solved.
`NonLocal (...}`	Defines a subregion and internal submesh near a Schottky barrier or heterointerface where barrier tunneling models are applied.
`MetalConductivity`	Switches off the conductivity of metals (see Section 4.2.5 on page 15.134), but the thermal conductivity is simulated according to the thermodynamic model.

For most problems, Newton iterations converge best with full derivatives. Furthermore, for small-signal analysis, and noise and fluctuation analysis, using full derivatives is mandatory. Therefore, by default, DESSIS takes full derivatives into account. For rare occasions where omission of derivatives improves convergence or performance significantly, use the keywords `-AvalDerivatives` and `-Derivatives` to switch off mobility and avalanche derivatives. The derivatives are usually computed analytically, but a numeric computation can be used by specifying `Numerically`. Table 15.28 summarizes the different derivative keywords.

Table 15.28 Math parameters for derivatives

Parameter	Description
`-AvalDerivatives`	Switches off the analytic derivatives of the avalanche terms (switched on by default).
`-Derivatives`	Switches off the analytic derivatives of the mobility and avalanche terms (switched on by default).
`Numerically (<equation-pair>, <equation-pair>,...)`	Switches on numeric derivatives, that is, the Jacobian is determined using a finite-difference scheme. The optional equation list can be used to restrict the calculation of the numeric Jacobian to specific equations. This is recommended only for debugging purposes because this causes the simulation to run very slowly. (It is switched off by default and does not work with the method `Blocked`.)

In some very important cases, DESSIS allows the use of different discretization approaches, which are controlled by the keywords in Table 15.29 on page 15.75.

The option `-NewDiscretization` switches on an old discretization scheme for the transport equations.

NOTE The option `-NewDiscretization` will be withdrawn in a future DESSIS release. Its use is discouraged except when using models that explicitly require it.

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

The electrostatic potential can be computed from an arbitrary reference potential level ψ_{ref}. In DESSIS, ψ_{ref} is computed from the vacuum level, using the following rules:

- If there is silicon in any simulated semiconductor structure, the intrinsic Fermi level of silicon is selected as reference, $\psi_{\mathrm{ref}} = \Phi_{\mathrm{intr}}(\mathrm{Si})$.
- Otherwise, if any simulated device structure contains GaAs, $\psi_{\mathrm{ref}} = \Phi_{\mathrm{intr}}(\mathrm{GaAs})$.
- In all other cases, DESSIS selects the material with the smallest band gap (assuming a mole fraction of 0) and takes the value of its intrinsic Fermi level as ψ_{ref}.

The keywords `DirectCurrentComputationAtContact` and `CurrentWeighting` control current output. By default, the current is computed using integration over doping wells. `CurrentWeighting` modifies this method to minimize error contributions. In the `DirectCurrent` option, the current is computed directly, without any additional integration.

Table 15.29 Math parameters for discretization methods

Parameter	Description
`CurrentWeighting`	Contact currents are computed using an optimal weighting scheme (it is switched off, by default).
`DirectCurrentComputationAtContact`	Contact currents are computed directly, using only contact nodes and their neighbors (it is switched off, by default).
`EvEpara`	This option provides accurate and reliable calculations for models that depend on the parallel electric field, such as avalanche generation and high field velocity saturation. It is switched on, by default. Alternatively, computations can be based on the averaging field-dependent mobility or avalanche generation rate over elements. This approach is fast for simple situations, but does not guarantee high accuracy and fast convergence rates of the Newton iteration process. To switch manually to such a mode, specify the keyword `-EvEpara` to switch off `EvEpara`.
`-NewDiscretization`	Switches back to the obsolete discretization scheme for continuity equations, lattice temperature equations, and carrier temperature equations.
`-ConstRefPot`	Selects a position-dependent ψ_{ref}, which is the local value of the intrinsic Fermi potential.
`ConstRefPot=<value>`	Specifies a user-defined value for ψ_{ref}.
`EnormalInterface`	Specifies the interface for electric field computation normal to that interface.

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

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

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

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

2.10.2 Math parameters for nonlinear iterations convergence control

During a `Solve` statement, DESSIS tries to determine the value of an equation variable x, such that the computed update Δx (after k-th nonlinear iteration) is small enough:

$$\frac{\left|\frac{\Delta x}{x^*}\right|}{\varepsilon_R\left|\frac{x}{x^*}\right| + \varepsilon_A} < 1 \qquad (15.9)$$

where:

$$\varepsilon_R = 10^{-\text{Digits}} \qquad (15.10)$$

and x^* is a scaling constant.

NOTE The condition (Eq. 15.9) only holds for a scalar equation. It can be generalized in the case of a vector of unknowns (see Section 32.5.1 on page 15.529).

It is clear that (Eq. 15.9) is equivalent to:

$$\frac{|\Delta x|}{|x| + x_{\text{ref}}} < \varepsilon_R \qquad (15.11)$$

where:

$$x_{\text{ref}} = \frac{\varepsilon_A}{\varepsilon_R}x^* \qquad (15.12)$$

By default, DESSIS uses the condition (Eq. 15.11), but the keyword `-RelErrControl` can be used to switch to (Eq. 15.9). For large values of x ($|x| \to \infty$), the conditions (Eq. 15.9) and (Eq. 15.11) are equivalent to the relative error criterion:

$$\frac{|\Delta x|}{|x|} < \varepsilon_R \qquad (15.13)$$

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

$$\left|\frac{\Delta x}{x^*}\right| < \varepsilon_A \quad \text{or} \quad |\Delta x| < x_{\text{ref}} \cdot \varepsilon_R \qquad (15.14)$$

respectively. DESSIS uses the expressions (Eq. 15.9) and (Eq. 15.11) to ensure a smooth transition between absolute and relative error control.

If `-RelErrControl` has been specified to disable relative error control, the user must specify the value of `Digits` to change the default value of the relative error. DESSIS uses (Eq. 15.10) to define ε_R internally. In absolute error specifications, the values of ε_A and x_{ref} can be defined independently for each equation variable.

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

Table 15.30 summarizes the available keywords for error control during nonlinear iterations. Table 15.31 lists the default values for the error control criteria.

Table 15.30 Math parameters for nonlinear iterations error control

Parameter	Description
`Digits = <float>`	Relative error convergence criterion. `Digits` approximates the number of digits of accuracy to which an equation must be solved before being considered to have converged. When either (Eq. 15.9) or (Eq. 15.11) is met, DESSIS assumes the equation is solved. Default is 5.
`Error(<equation-variable>) = <float>`	Defines the value of ε_A in (Eq. 15.9).
`ErrRef(<equation-variable>) = <float>`	Defines the value of x_{ref} in (Eq. 15.11).
`RelErrControl | -RelErrControl`	Specifies that the unscaled expression (Eq. 15.11) is used for error control. The default is `RelErrControl`.

Table 15.31 Equation variables and their default errors

<equation-variable>	Error criterion for equations	Default of error (without RelErrControl)	Default of ErrRef (with RelErrControl)
`Poisson`	Poisson	1.0×10^{-3}	0.0258 V
`Electron`	Electron continuity	1.0×10^{-5}	10^{10} cm^{-3}
`Hole`	Hole continuity	1.0×10^{-5}	10^{10} cm^{-3}
`Temperature`	Temperature	1.0×10^{-3}	300 K
`ElectronTemperature`	Electron temperature	1.0×10^{-4}	300 K
`HoleTemperature`	Hole temperature	1.0×10^{-4}	300 K
`Contact`	Contact current	1.0×10^{-3}	0.0258 V
`Circuit`	Circuit equations	1.0×10^{-3}	0.0258 V

2.10.3 Math parameters for transient analysis

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

To activate a particular transient method, `Transient=<transient-method>` must be specified, where `<transient-method>` can be `TRBDF` or `BE`. Together with error control of nonlinear equations convergence (which is needed for both DC and transient analysis), time-step control is necessary during transient simulation (see Section 32.4 on page 15.527, where equations and criteria for time-step control are described).

To activate time-step control, `CheckTransientError` must be specified (it is switched off by default, that is, time-step depends only on convergence of Newton iterations). For time-step control, DESSIS uses a separate set of criteria, but as with Newton iterations, the control of both relative and absolute errors is performed. Relative transient error $\varepsilon_{R,\,tr}$ is defined similarly to ε_R in (Eq. 15.10):

$$\varepsilon_{R,\,tr} = 10^{-\mathrm{Digits,tr}} \tag{15.15}$$

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

$$x_{\mathrm{ref,tr}} = \frac{\varepsilon_{A,\,tr}}{\varepsilon_{R,\,tr}} x^* \tag{15.16}$$

is valid. The keyword `TransientDigits` must be used to specify relative error $\varepsilon_{R,\,tr}$ in transient simulations. An absolute error in time-step control can be specified by either the keyword `TransientError` ($\varepsilon_{A,\,tr}$) or `TransientErrRef` ($x_{\mathrm{ref,tr}}$), and their values can be defined independently for each equation variable. The same flag as in Newton iteration control, `RelErrControl`, is used to switch to unscaled (`TransientErrRef`) absolute error specification. Table 15.32 summarizes the keywords available in the `Math` section for transient analysis.

Table 15.32 Math parameters for transient analysis

Parameter	**Description**
`Transient = <transient-method>`	Defines the transient discretization scheme. Available options for transient method are `TRBDF` and `BE`. The default is `TRBDF`.
`TransientDigits = <float>`	Defines relative error convergence criterion $\varepsilon_{R,\,tr}$ in time-step control ((Eq. 15.15)). The default is `TransientDigits = 3`.
`TransientError(<equation-variable>) = <float>`	Defines the value of $\varepsilon_{A,\,tr}$ in time-step control.
`TransientErrRef(<equation-variable>) = <float>`	Defines the value of $x_{\mathrm{ref,tr}}$ in time-step control.
`CheckTransientError | NoCheckTransientError`	Enables or disables all error controls through the transient integration method. If disabled, error control is only managed by the convergence property of the inner solve method. Default is `NoCheckTransientError`.
`TrStepRejectionFactor`	Defines the value of f_{rej} factor (see Section 32.4.3 on page 15.529).

NOTE All transient parameters in the `Math` section, except `Transient = <transient-method>`, can be overwritten in the `Transient` command of the `Solve` section (see Section 2.9.4 on page 15.62).

2.10.4 Solver-oriented Math keywords

The Math parameters to the solution algorithms are device independent and must only appear in the base Math section. These can be grouped by solver type. The control parameters for the linear solvers are Method and SubMethod. The keyword Method selects the linear solver to be used, and the keyword SubMethod selects the inner method for block-decomposition methods (see Table 15.33).

Table 15.33 Math parameters for linear solver

Method = *<solver>* Selects the linear solver to be used in the Coupled command.	*<solver>*
	Blocked: Selects a block decomposition solver (default). Extra parameter SubMethod specifies the inner solver to be used.
	ILS: Selects the (parallel) iterative linear solver ILS.
	ParDiSo: Selects the (parallel) supernodal direct solver.
	Slip: Selects the Slip iterative solver.
	Super: Selects the direct supernodal solver.
	UMF: Selects the UMFPACK solver.
SubMethod = *<solver>* Specifies the solver to use in the inner solver.	*<solver>*
	ILS: Selects the (parallel) iterative linear solver ILS.
	ParDiSo: Selects the (parallel) direct supernodal solver.
	Slip: Selects the Slip iterative solver.
	Super: Selects the direct supernodal solver.
	UMF: Selects the UMFPACK solver.
ExitOnFailure	Selects termination of the simulation as soon as a Solve command fails.

The UMF solver recognizes the parameters shown in Solvers, Chapter 4 on page 19.13. These parameters can be specified inside the Math section as follows:

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

The Coupled command is sensitive to the Math parameters Iterations, LineSearchDamping, NotDamped, RhsMin, and RhsFactor. Iterations, LineSearchDamping, and NotDamped in the Math section set the defaults to the equivalent parameters as in the Coupled command.

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

Table 15.34 Math parameters for coupled solver

Keyword	Description
`Iterations = <int>`	Maximum number of iterations. If the equation being solved has not converged after this number of iterations, the solution step stops and the next `Solve` command starts (as determined in the `Solve` statement) (default is `Iterations = 50`). If the equation being solved is converging quadratically, the number of iterations is automatically extended beyond `Iterations`. If the user defines `Iterations = 0`, however, only one iteration is performed, regardless of the convergence behavior.
`NotDamped = <int>`	Number of iterations in each Newton iteration in which the RHS-norm is allowed to increase (default is `NotDamped = 1000`) without Bank–Rose damping being activated.
`LineSearchDamping = <float>`	Sets the smallest allowed damping coefficient for line search damping. The default is `LineSearchDamping=1`.
`Traps(Damping = <float>)`	Sets damping for `Traps` (see Chapter 10 on page 15.225) for the nonlinear Poisson equation. Larger values increase damping; a value of `0` disables damping. The default is `Traps(Damping=10)`.
`RhsMin = <float>`	Minimum size of L_2-norm of the RHS in each Newton iteration (default `RhsMin` $= 10^{-5}$).
`RhsMax = <float>`	Maximum size of L_2-norm of the RHS in each Newton iteration (default `RhsMax` $= 10^{15}$). This parameter is only used during transient simulations.
`RhsFactor = <float>`	Maximum increase of the L_2-norm of the RHS between Newton iterations (default `RhsFactor` $= 10^{10}$).

The `Quasitationary` and `Transient` commands are sensitive to the `Math` parameters `Extrapolate`, `Smooth`, and `BreakAtIonIntegral`. The keyword `Extrapolate` allows the previous solutions of the `Quasistationary` or `Transient` to be used to extrapolate a new solution at each step. When the keyword `Smooth` is specified, the solution is always evaluated twice: with simplified physics and, then, with all physical models. This method is sometimes useful in difficult situations. `BreakAtIonIntegral` forces a `Quasistationary` to stop when the largest ionization integral is greater than one.

Table 15.35 Math parameters for quasistationary and transient solvers

Parameter	Description
`Extrapolate`	Extrapolation is used in quasistationary and transient simulations. The variables for the initial solution for a given time step or quasistationary step are computed using an extrapolation from the previous two steps. (Extrapolation is off by default.)
`Smooth`	Smoothing iterations that keep mobility and recombination data from the previous step to obtain better initial conditions for extreme nonlinear iterations. (Smoothing is off by default.)
`BreakAtIonIntegral`	Used to terminate the quasistationary simulation when the largest ionization integral is greater than one. The complete syntax of this keyword is `BreakAtIonIntegral (number value)` where a quasistationary simulation finishes if the `number` ionization integral is greater than `value`, where the ionization integrals are ordered with respect to decreasing value.

NOTE Another `Math` section parameter, the keyword `NoAutomaticCircuitContact`, is used in the context of mixed circuit and device systems (see Chapter 3 on page 15.101).

2.10.5 Break criteria

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

The following values can be monitored during a simulation:

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

It is possible to specify values to a lower bound and an upper bound. Similarly, a bound can be specified on the absolute value.

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

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

In this example, the stopping criterion is met if the absolute value of the inner voltage at the drain exceeds 10 V. In addition, DESSIS terminates the simulation if the source current is less than –0.001 A/µm or greater than 0.002 A/µm.

NOTE The unit A/µm is valid for 2D devices; the unit A/µm^2 is valid for 1D devices; and the unit A is valid for 3D devices.

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

```
Math (material = "Silicon") {
   ...
   BreakCriteria {
      LatticeTemperature (maxval = 1000)
      CurrentDensity (absval = 1e7)
   }
   ...
}
```

```
Math (region = "Region.1") {
   ...
   BreakCriteria {
      ElectricField (maxval = 1e6)
   }
   ...
}
```

DESSIS terminates the simulation if the lattice temperature in silicon exceeds 1000 K, the current density in silicon exceeds 10^7 A/cm^2, or the electrical field in the region `Region.1` exceeds 10^6 V/cm.

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

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

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

2.10.6 Parallelization

DESSIS uses thread parallelism to accelerate simulations on shared memory computers. The following computations have been parallelized:

- Mobility
- Avalanche
- Current density
- Energy flux

The required number of threads and the stack size per thread can be specified in the global `Math` section of the DESSIS command file:

```
Math {
   number_of_threads = number of threads
   stacksize = stacksize in bytes
}
```

Alternatively, the following UNIX environment variables are recognized:

```
DESSIS_NUMBER_OF_THREADS, ISE_NUMBER_OF_THREADS
DESSIS_STACKSIZE, ISE_STACKSIZE
```

By default, DESSIS uses only one thread and the stack size is 1 MB. For most simulations, the default stack size is adequate. The following restriction applies:

- On Sun and Linux, it is possible to run PARDISO™ on multiple processors. Unfortunately, the additional threads created by PARDISO are not suspended after the solution of the linear system. As a result, the PARDISO threads and DESSIS threads may compete for the same processors, resulting in a performance loss.

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

- Speedups are only obtained for sufficiently large problems. As a general rule, the device grid should have at least 5000 vertices. Three-dimensional problems are good candidates for parallelization.
- It is sensible to run a parallel DESSIS job on an empty computer. As soon as multiple jobs compete for processors, performance decreases significantly.
- Use the keyword `wallclock` in the `Math` section of the DESSIS command file to display wall clock times rather than CPU times.
- The parallel execution of PARDISO produces different rounding errors. Therefore, the number of Newton iterations may change.
- On HP-UX 11, it is recommended to set the UNIX environment variable `MP_GANG=OFF`. This disables gang scheduling.

2.10.7 Nonlocal line meshes

Nonlocal line meshes are one dimensional, special-purpose meshes that DESSIS needs to implement one-dimensional, nonlocal physical models. A nonlocal line mesh consists of nonlocal lines, each of which connects a vertex of the normal mesh to the interface or the contact for which the nonlocal line mesh is constructed. Each nonlocal line mesh is subdivided into nonlocal mesh points, to allow for the discretization of the equations that constitute the physical models.

The 1D Schrödinger equation (see Section 7.3 on page 15.167) and the nonlocal tunneling model (see Section 16.4 on page 15.306) use nonlocal line meshes. The documentation of these models introduces the use of nonlocal line meshes in the context of the particular model and is restricted to typical cases. This section describes the construction of nonlocal line meshes in detail.

2.10.7.1 Specifying nonlocal line meshes

Nonlocal line meshes are specified by the `Nonlocal` keyword in an interface-specific or a contact-specific `Math` section. The options of the keyword `NonLocal` control the construction of the nonlocal line mesh. DESSIS takes some of the options (`Length`, `Permeation`, `Direction`, `MaxAngle`, and `-Outside`) from the `Math` section specific to the interface or contact for which a nonlocal line mesh is constructed, and other options (`-Endpoint`, `-Transparent`, `-Permeable`, and `-Refine`) from the `Math` sections specific to materials or regions. For a summary of available options, see Table 15.36 on page 15.84.

For example, with:

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

DESSIS constructs nonlocal lines for semiconductor vertices up to a distance of 5 nm from the `Gate` electrode.

Table 15.36 Options to NonLocal to control the construction of a nonlocal mesh

Keyword	Application	Description
`Length=<len>`	Interface or contact	Sets the distance from the interface or contact up to which nonlocal mesh lines are constructed; `<len>` in centimeters.
`Permeation=<len>`	Interface or contact	Length [cm] by which nonlocal mesh lines are extended across the interface or contact (default is zero).
`Direction=<vector>`	Interface or contact	Specifies a direction as a 3D vector. If nonzero, the construction of nonlocal mesh lines with a direction more than `MaxAngle` degrees different from the vector is suppressed (default is `(0 0 0)`).
`MaxAngle=<val>`	Interface or contact	Suppresses construction of nonlocal mesh lines that enclose an angle of more than `<val>` degrees with the vector specified by `Direction` (default is 180).
`-Outside`	Interface or contact	Prohibits nonlocal mesh lines leaving the device (default is `Outside`).
`-Endpoint`	Region or material	Prohibits construction of nonlocal lines that end in the region. Default is `Endpoint` for semiconductor regions. For other regions, DESSIS uses `-Endpoint` always, and ignores `Endpoint` options in the command file.
`-Transparent`	Region or material	Prohibits nonlocal lines crossing the region (default is `Transparent`).
`-Refined`	Region or material	Disables the refinement of nonlocal mesh in the interior of the region (default is `Refined`).
`-Permeable`	Region or material	Inhibits extension of nonlocal lines (as specified by the `Permeation` parameter) into or across the region (default is `Permeable`).

2.10.7.2 Visualizing nonlocal line meshes

DESSIS can visualize the nonlocal line meshes it constructs. This feature is used to verify that the nonlocal line mesh constructed is the one actually intended. For visualizing data defined on nonlocal line meshes, see Section 2.10.7.3 on page 15.85.

To enable visualization of nonlocal line meshes, use the keyword `NonLocal` in the `Plot` section (see Section 2.6 on page 15.52). The keyword causes DESSIS to write two vector fields to the plot file that represent the nonlocal line meshes constructed in the device.

For each vertex (of the normal mesh) for which a nonlocal mesh line exists, the first vector field `NonLocalDirection` contains a vector that points from the vertex to the end of the nonlocal mesh line in the direction of the interface or contact for which the nonlocal mesh line was constructed. The vector in the second field `NonLocalBackDirection` points from the vertex to the other end of the nonlocal mesh line. The unit of both vectors is μm .

For vertices for which no nonlocal mesh line exists, both vectors are zero. For vertices for which more than one nonlocal mesh line exists, DESSIS plots the vectors for one of these lines.

2.10.7.3 Visualizing data defined on nonlocal line meshes

To visualize data defined on nonlocal line meshes:

- In the `File` section (see Section 2.2 on page 15.38), specify a file name using the `NonLocalPlot` keyword.
- On the top level of the command file, specify a `NonLocalPlot` section. There, `NonLocalPlot` is followed by a list of coordinates in parentheses and a list of datasets in braces.

DESSIS writes nonlocal plots at the same time it writes normal plots. Nonlocal plot files have the extension `.plt`.

DESSIS picks nonlocal mesh lines close to the coordinates specified in the `NonLocalPlot` section for output. The datasets given in the `NonLocalPlot` section are the datasets that can be used in the `Plot` section (see Section 2.6 on page 15.52). `NonLocalPlot` does not support the `/Vector` option. Additionally, the Schrödinger equation provides special-purpose datasets available only for `NonLocalPlot` (see Section 7.3.4 on page 15.170).

In addition to the datasets explicitly specified, DESSIS automatically includes the `Distance` dataset in the output. It provides the coordinate along the nonlocal mesh line. Data in the `Distance` dataset is measured in μm. The interface or contact for which a nonlocal mesh line was constructed is located at zero, and its mesh vertex is located at positive coordinates. For example:

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

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

2.10.7.4 Constructing nonlocal line mesh

Figure 15.21 on page 15.86 shows four examples of nonlocal mesh lines (denoted by the letters A, B, C, and D) connected to a contact or an interface. For simplification, the figure shows a tensorial patch of the normal mesh and a planar contact. (This is not required for the nonlocal line mesh construction to work.)

The construction in Figure 15.21 assumes that regions R1, R2, and R3 are semiconductor regions, that none of the interfaces between those regions is treated as a heterointerface, and that all options for nonlocal mesh generation have their default values. The circles denote nonlocal mesh points. The nonlocal mesh points are not part of the normal mesh that DESSIS uses; they are part of the special-purpose nonlocal mesh only. However, the nonlocal mesh points with index ‘0’ (A_0, B_0, C_0, and D_0) coincide with normal vertices, namely, with the vertex for which their nonlocal line was constructed.

The nonlocal mesh points with the indices ‘–1’ and ‘1’ are the intersections of the nonlocal mesh line with the box (see Section 32.1 on page 15.519) associated to its vertex (Figure 15.21 does not show the boxes themselves). DESSIS generates further nonlocal mesh points wherever a nonlocal mesh line crosses the boundary of elements. This approach guarantees that DESSIS can interpolate data from the normal mesh to the nonlocal mesh lines with optimal accuracy.

Mesh vertices on heterointerfaces receive special treatment. DESSIS constructs separate nonlocal lines for each region at the heterointerface. For example, if the interface between R2 and R3 in Figure 15.21 is a heterointerface, DESSIS constructs two nonlocal lines for the vertex A_0.

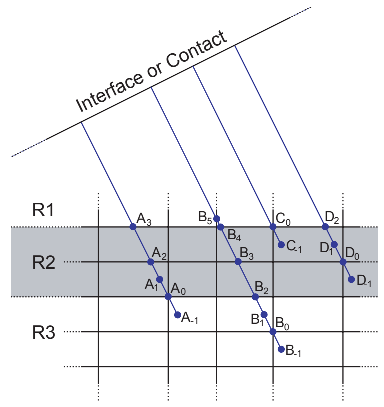


Figure 15.21 Construction of a nonlocal mesh

The nonlocal mesh line that links a vertex to a contact or interface does so on the geometrically shortest path. The parameter `Length` determines the maximum distance of the vertex to the contact; therefore, all points indexed by '0' in the figure are not further away from the contact than this distance. However, the distance of the vertices indexed by '–1' from the interface or contact may be larger. `Length` is a parameter specific to the interface or contact for which the nonlocal mesh is constructed.

The property that nonlocal mesh lines connect a vertex to a contact or an interface on the geometrically shortest path is fundamental. If any of the other rules described in this section inhibits the construction of a nonlocal mesh line for this path, but a longer connection obeys all these restrictions, DESSIS still does not use this connection to construct an alternative nonlocal line.

When the device is not convex, the shortest connection from a mesh vertex to an interface or a contact can be partly or entirely outside the device. To suppress the construction of nonlocal mesh lines that leave the device, specify `-Outside` for the interface or contact in question.

If the option `-Endpoint` is present for a region or the region is an insulator, DESSIS does not construct nonlocal mesh lines that end in this region. For example, if `-Endpoint` is specified for region R2 in Figure 15.21, DESSIS does not construct line D. Line C will be shorter and will end at C_0 rather than C_{-1}. In addition, DESSIS omits point A_1.

If the `-Transparent` option is present for a region, DESSIS does not construct nonlocal mesh lines that cross it. For example, if `-Transparent` is specified for region R2 in Figure 15.21, DESSIS constructs only line C; all other lines must cross R2 at least partially.

The `Direction` parameter specifies a direction that the nonlocal mesh lines approximately should have. Nonlocal mesh lines with directions that deviate from the specified direction by an angle greater than `MaxAngle` are suppressed. If `Direction` is the zero vector or `MaxAngle` exceeds 90 (this is the default), nonlocal lines can have any direction. For example, `Direction=(1 0 0)` and `MaxAngle=5` allow only nonlocal mesh lines that run horizontally, with a tolerance of 5^o. In Figure 15.21, all lines would be suppressed.

The `-Refined` option for a region suppresses the generation of nonlocal mesh points at element boundaries in the interior of that region. For example, if `-Refined` is specified for region R2 in Figure 15.21, DESSIS does not generate the points A_2 and B_3. However, DESSIS always generates nonlocal mesh points at region interfaces. For example, if `-Refined` is specified for both R2 and R3 in Figure 15.21, DESSIS generates B_2 nevertheless.

2.10.7.5 Special handling of 1D Schrödinger equation

For performance reasons, DESSIS solves the 1D Schrödinger equation (see Section 7.3 on page 15.167) only on a reduced subset of nonlocal mesh lines that still cover all vertices of the normal mesh for which nonlocal mesh lines are constructed according to the rules outlined above.

To avoid artificial geometric quantization, DESSIS extends nonlocal mesh lines used for the 1D Schrödinger equation that are shorter than `Length` beyond the mesh points indexed by '–1' in Figure 15.21 on page 15.86 to reach full length. Furthermore, the parameter `Permeation` specifies a length by which DESSIS extends the nonlocal mesh lines at the other end, across the interface. For both ends, DESSIS never extends the lines outside the device or into regions flagged by the `-Permeable` option. The extension is not affected by the `Transparent` and `Endpoint` options. The extensions of the nonlocal mesh lines are ignored by the nonlocal tunneling model (see Section 16.4 on page 15.306).

For nonlocal mesh line segments in regions not marked by `-Refined` and `-Endpoint`, DESSIS computes the intersections with the boxes of the normal mesh. These intersection points form a refinement of the nonlocal mesh lines in addition to what Figure 15.21 shows. DESSIS needs this information to interpolate results from the 1D Schrödinger equation back to the normal mesh. Therefore, in regions where `-Refined` or `-Endpoint` is specified, 1D Schrödinger density corrections are not available, even when the regions are covered by nonlocal mesh lines.

2.10.7.6 Special handling of nonlocal tunneling model

The nonlocal mesh points with the indices '–1' and '1' in Figure 15.21 are the intersections of the nonlocal mesh line with the boxes (see Section 32.1 on page 15.519). The nonlocal tunneling model (see Section 16.4) uses these two points to limit the spatial range of the integrations DESSIS must perform to compute the contribution to tunneling that comes from the particular vertex.

If A_0 in Figure 15.21 was on a heterointerface, two nonlocal mesh lines would be constructed for it. The spatial integrations for the first nonlocal mesh line go from A_1 to A_0, and the integrations for the second nonlocal mesh line go from A_0 to A_{-1}.

Conversely, if the option `-Endpoint` is specified for region R2 in Figure 15.21, DESSIS omits point A_1 and extends the integrations for line A from A_3 to A_{-1} in order to pick up the Fowler–Nordheim tunneling current that may enter region R2.

For nonlocal line meshes used only for nonlocal tunneling, when `Permeation` zero, DESSIS assumes that the nonlocal lines it constructs for an interface cross the interface. Therefore, if `-Endpoint` is specified for one of the regions that forms the interface for which the nonlocal mesh is constructed, DESSIS constructs only mesh lines that go through that region and, therefore, end in the facing region. For example, if `-Endpoint` is specified for the region above the interface shown in Figure 15.21, DESSIS does not construct any of the four nonlocal mesh lines. If `-Endpoint` is specified for the region below, DESSIS constructs all of the nonlocal mesh lines, provided this region does not coincide with R1, R2, or R3.

2.10.7.7 Performance suggestions

To limit the negative performance impact of the nonlocal tunneling model, it is important to limit the number of nonlocal mesh lines. To this end, most importantly, select `Length` to be only as long as necessary. The option `-Endpoint` can be used to suppress the construction of lines to regions for which you know, in advance, that will not receive much tunneling current. The option `-Transparent` allows users to neglect tunneling through materials with comparatively high tunneling barrier, for example, oxides near a Schottky contact or heterointerface for which nonlocal tunneling has been activated.

Another use of the option `-Transparent` is at heterointerfaces, where there is no tunneling to the side of the material with the lower band edge (as there is no barrier to tunnel through). To restrict the construction of nonlocal mesh lines to lines that go through the larger band-edge material, declare the lower band-edge material as not transparent by using `-Transparent`.

The option `-Refined` does not reduce the number of nonlocal lines, but it can reduce the degree of nonlocality. For example, if `-Refined` is specified to region R2 in Figure 15.21 on page 15.86, the tunneling current for line B is independent of the solution variables in the interior of region R2. The option `-Refined` is most useful for insulator regions, where the band-edge profile is approximately linear.

2.10.8 Monitoring convergence behavior

When DESSIS has convergence problems, it can be helpful to know in which parts of the device and for which equations the errors are particularly large. With this information, it is easier to make adjustments to the mesh or the models used, to improve convergence.

DESSIS can print the locations in the device where the largest errors occur (see Section 2.10.8.1). This provides limited information and has negligible performance impact. DESSIS can also plot solution error information for the entire device after each Newton step (see Section 2.10.8.2). This information is comprehensive, but can generate many files and can take significant time to write.

Both approaches provide access to DESSIS internal data. Therefore, in both cases, the output is implementation dependent. Its proper interpretation can change between different DESSIS releases.

2.10.8.1 CNormPrint

To obtain basic error information, specify the `CNormPrint` keyword in the global `Math` section. Then, after each Newton step and for each equation solved, DESSIS prints to the standard output:

- The largest error according to (Eq. 15.11) that occurs anywhere in the device for the equation.
- The vertex where the largest error occurs.
- The coordinates of the vertex.
- The current value of the solution variable for that vertex.

2.10.8.2 NewtonPlot

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

- Use the `NewtonPlot` keyword in the `File` section (see Section 2.2 on page 15.38) to specify a file name for the plot. This name can contain up to two C-style integer format specifiers (for example, `%d`). If present, for the file name generation, the first one is replaced by the iteration number in the current Newton step and the second, by the number of Newton steps in the simulation so far. DESSIS does not enforce any particular file name extension, but prepends the device instance name to the file name in mixed mode.

- Optionally, specify the data for output as options to the `NewtonPlot` keyword in the `Math` section. By default, DESSIS writes the current values of the solution variables only. Table 15.37 lists the available options.

Table 15.37 Options to NewtonPlot

Option	Description
`Plot`	Writes all data specified in the `Plot` section (see Section 2.6 on page 15.52).
`Residual`	Writes the residuals (right-hand sides) of all equations.
`Update`	Writes the updates of all solution variables from the previous step.

2.11 Thermodynamic simulations

Apart from the standard drift-diffusion model, DESSIS can simulate self-heating effects by using the thermodynamic model. A description of the physical background of the thermodynamic model and its formulation (differential equations, boundary conditions, parameters) is in Section 4.2.3 on page 15.128. When the electrostatic potential, and the electron and hole densities (or quasi-Fermi potentials) have been determined, the amount of heat generation in the device can be determined and plotted.

To simulate self-heating effects on the temperature distribution and, consequently, the effects of the nonuniform temperature distribution on the electrical characteristics, an additional equation is solved whose keyword is `Temperature`. The `Solve` statement in this case is:

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

It is recommended that the coupled mode be used for nonisothermal simulation. The plugin mode can also be used in nonisothermal simulations, particularly if the coupling between drift-diffusion equations and lattice temperature equations is not very strong.

2.11.1 Nonisothermal simulation recommendations

In nonisothermal simulations, it is recommended to always include as much of the substrate die as possible, as well as packaging materials and heat sinks. Simulations of the thermal boundary conditions make the choice of the thermal surface resistance (or conductance) less critical.

Furthermore, since the area of thermal interest is sometimes orders of magnitude greater than the area of electrical interest, confining nonisothermal simulations to the electrical simulation domain is problematic. This is because the accuracy of the maximum, minimum, and average temperatures, and the temperature distribution are greatly compromised by unnatural and unrealistic thermal boundary conditions.

2.12 Hydrodynamic simulations

Another transport model in DESSIS is the hydrodynamic model. A description of the physical background of the hydrodynamic model is in Section 4.2.4 on page 15.130. To activate the hydrodynamic simulation mode, the user must include the keyword `Hydrodynamic` in the `Physics` section of the DESSIS input file.

The electron, hole, and lattice temperatures (T_n, T_p, and T_L) are solved by specifying the keywords `ElectronTemperature`, `HoleTemperature`, and `LatticeTemperature` (or `Temperature`) respectively, in the `Solve` section of the DESSIS input file.

DESSIS allows both coupled and plugin hydrodynamic simulations. A Scharfetter–Gummel-like discretization scheme is used for the discretization of the continuity equations and energy balance equations. Different sets of equations can be included in the model. If only the equation for one carrier temperature is to be solved, to ensure a consistent model, it is recommended that it is specified explicitly by using it as a parameter to the keyword `Hydrodynamic` (that is, `Hydrodynamic(eTemp)` or `Hydrodynamic(hTemp)`). `Hydrodynamic` without a parameter must be specified if equations for both carrier temperatures are to be included in the model.

Equations can be solved in arbitrary plugin-coupled combinations. The syntax for a pure coupled `Solve` statement is, for example:

```
Solve {
   Coupled {Poisson Electron Hole ElectronTemperature
            HoleTemperature LatticeTemperature}
}
```

A plugin iteration for a system of four equations (Poisson, electron, hole, and electron temperature), for a suggested strategy [8], can be written as:

```
Solve {
   Plugin {
      Coupled {Poisson Electron Hole}
      Coupled (Iteration=0) {Electron ElectronTemperature}
   }
}
```

For the conditions remote from breakdown, the convergence rate for the above plugin strategy can be almost as fast as for coupled iterations [9][10].

For physical models, it is recommended that a carrier temperature–dependent model is selected for the high field saturation of the mobility, and not the usual Canali field-dependent model.

This can be performed by using the construct `HighFieldSaturation(CarrierTempDrive)` as an option to `Mobility` in the `Physics` section of the DESSIS input file (see Section 8.8 on page 15.193). Analogously, it is possible to select a local carrier temperature–dependent, impact ionization model using the construct `Avalanche(CarrierTempDrive)` as an option to the keyword `Recombination` (see Section 9.9 on page 15.213). To use different driving forces for electrons and holes, the following syntax can be used:

```
eHighFieldSaturation(<drivingforce>), hHighFieldSaturation(<driving force>),
eAvalanche(<model type> <driving force>), and
hAvalanche(<model type> <driving force>).
```

If the keyword `Save` is specified in the `File` section of the DESSIS input file, the three temperatures T_n, T_p, and T_L are stored in the `.sav` file with the electrical state variables *y*, *n*, and *p*. These values are used as an initial guess if `Load` is specified.

2.13 Parameter and model specification

2.13.1 Region and material parameter specification

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

2.13.2 Generating a copy of parameter file

To redefine a parameter value for a particular material, a copy of the default parameter file must be created. To do this, the command `dessis -P` prints the parameter file for silicon, with insulator properties. Table 15.38 lists the principal options for the command `dessis -P`.

Table 15.38 Principal options for generating parameter file

Option	Description
`-P:All`	Prints a copy of the parameter file for all materials. Materials are taken from the file `datexcode.txt`. The first section of the parameter file does not contain material reference. It includes all models and parameters available inside DESSIS. Sections with references to specific materials include only the default models defined for the particular material.
`-P:Material`	Prints model parameters for the specified material.
`-P filename`	Prints model parameters for materials and interfaces used in the file name.

NOTE One restriction to the use of a parameter file is that only one file is permitted in any simulation. Therefore, the parameter sections for all the various materials and regions must be concatenated into a single `.par` file, which can be unwieldy and lead to errors. It is more convenient to create a 'library' of parameter files (see Section 2.13.5 on page 15.92).

2.13.3 Changing parameter values in parameter file

The section of the parameter file for predefined materials is:

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

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

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

For region-specific parameter specifications, the syntax is:

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

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

2.13.4 Hierarchy of parameter specifications

To avoid confusion, it is important to understand the hierarchy of region and material parameter specifications. A methodology used for the specification of a parameter is similar to the hierarchy of physical models described in Section 2.5.4 on page 15.48:

- Default parameters are defined in the parameter file section for the particular material.

NOTE The parameters of a particular model may not be defined for a given material. However, if this model is used and parameters are not redefined in the appropriate region or material section of the parameter file, parameters of the default material (silicon) are used.

- The material section of the parameter file allows the user to overwrite default values for a particular material.
- The region section also overwrites the default values for the appropriate material. However, if for the region `reg` with material `mat`, both the region and material sections of the parameter file are defined, changes from the `mat` section are valid in all regions with this material except the region `reg`. In the region `reg` only, the changes from the region sections of the parameter file are applied.

2.13.5 Library of materials

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

```
$ISEROOT/tcad/$ISERELEASE/lib/dessis/MaterialDB
```

If a user-specific library is required or a group of users need to create a shared parameter library, it is necessary to create the library as a directory and define a new environment variable `DESSISDB` with a path to that directory. The library must contain a number of parameter files with each material having a separate parameter file, for example, `Silicon.par`, `GaAs.par`, and `Oxide.par`. The DESSIS command option `dessis -L` is used to create such files. Table 15.39 on page 15.93 lists the options of this command.

Table 15.39 Principal options for creating a library of materials

Option	Description
`-L:All`	Creates separate parameter files for all materials that DESSIS recognizes.
`-L:Material`	Creates a model parameter file for a specified material.
`-L <inputfile.cmd>`	DESSIS automatically examines the device structure to be simulated. It scans the `.grd` file specified in the `File` section of the input file `<inputfile.cmd>`. Separate parameter files for each material, interface, and electrode found in the `.grd` file are created, and a parameter file `dessis.par` is written.

For example, the simulation of the simple MOSFET structure from Section 1.4 on page 15.7 (`nmos_mdr.grd`, `nmos_mdr.dat`) consists of four gate oxide regions on a silicon region and four electrodes (`gate`, `drain`, `source`, and `substrate`).

The DESSIS command file `pp1_des.cmd` contains the following `File` section:

```
File {* input files:
   Grid = "nmos_mdr.grd"
   Doping = "nmos_mdr.dat"
   ...
}
```

By using the command:

```
dessis -L pp1_des.cmd
```

a parameter file is created in the current directory for each material, material interface, and electrode found in `nmos_mdr.grd`. In this example, the appropriate files are `Silicon.par`, `Oxide.par`, `Oxide%Silicon.par`, `Oxide%Oxide.par`, and `Electrode.par`.

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

```
Region = "Region0" { Insert = "Silicon.par" }
Region = "Region1" { Insert = "Oxide.par" }
Region = "Region3" { Insert = "Oxide.par" }
Region = "Region4" { Insert = "Oxide.par" }
Region = "Region5" { Insert = "Oxide.par" }
RegionInterface = "Region1/Region0" { Insert = "Oxide%Silicon.par" }
RegionInterface = "Region4/Region1" { Insert = "Oxide%Oxide.par" }
RegionInterface = "Region3/Region4" { Insert = "Oxide%Oxide.par" }
RegionInterface = "Region5/Region3" { Insert = "Oxide%Oxide.par" }
RegionInterface = "Region5/Region1" { Insert = "Oxide%Oxide.par" }
Electrode = "gate" { Insert = "Electrode.par" }
Electrode = "source" { Insert = "Electrode.par" }
Electrode = "drain" { Insert = "Electrode.par" }
Electrode = "substrate" { Insert = "Electrode.par" }
```

This `dessis.par` file can be renamed, but it is only used in the simulation if it is specified in the `File` section of the DESSIS command file:

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

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

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

NOTE The environment variable `DESSISDB` is also used to locate the file `Molefraction.txt` (see Section 2.13.5 on page 15.92).

NOTE The optical database file `optikdata` can be placed in the library, so that there is one location for all material parameters. The ability to define the environment variable `OPTIKDB` as in earlier DESSIS versions is still available (see Section 2.13.5).

2.13.6 Parameters of compound materials

A detailed parameter representation of compound materials in DESSIS is described in Section 18.6 on page 15.325. The command to print default parameters of such materials is the same as for regular monomaterials (see Section 2.13.2 on page 15.91), but for certain models (see Section 18.5 on page 15.324), it prints polynomial coefficients of approximations, which are dependent on a composition fraction.

Sometimes, it is difficult to analyze such coefficients to obtain a real value of physical models (for example, the band gap) for certain composition mole fractions. By using the command `dessis -M <inputfile.cmd>`, DESSIS creates a `dessis-M.par` file, which will contain regionwise parameters with only constant values (instead of the polynomial coefficients) for regions where the composition mole fraction is constant. For regions where the composition is not a constant, DESSIS prints default material parameters.

2.13.7 Undefined physical models

For a nonsilicon simulation, the default DESSIS behavior is to use silicon parameters for models that are not defined in a material used in the simulation. It is useful for noncritical models, but it can lead to confusion, for example, if the semiconductor band gap is not defined, DESSIS uses a silicon one. Therefore, DESSIS has a list of critical models and stops the simulation, with an error message, if these models are not defined.

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

Table 15.40 lists the critical models (with names from the DESSIS parameter file) with materials where these models are checked.

Table 15.40 List of critical models

Model	Insulator	Semiconductor	Conductor
`Auger`		x	
`Bandgap`		x	
`ConstantMobility (e/h)`		x	
`DopingDependence (e/h)`		x	
`DOSmass (e/h)`		x	
`Epsilon`	x	x	
`Kappa`	x	x	x
`RadiativeRecombination`		x	
`RefractiveIndex`	x	x	
`Scharfetter`		x	
`SchroedingerParameters`	x	x	

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

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

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

2.14 Material and doping specification

DESSIS supports all materials that are declared in the file `datexcodes.txt` (see Utilities, Chapter 2 on page 6.23).

By default, DESSIS supports all typical species of silicon technology (donors: As, P, Sb, and N, and acceptors: B and In) and allows user-defined ones (see Section 2.14.2 on page 15.98).

DESSIS loads doping distributions from the input doping file specified in the `File` section (see Section 2.2 on page 15.38) and reads the following datasets:

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

DESSIS supports the following rules of doping specification:

- DESSIS takes the net doping dataset from the file if this dataset is present. Otherwise, net doping is recomputed from the concentrations of the separate species.
- The same rule is applied for the total doping dataset, `Total`.

NOTE Total concentration, which originates from process simulators (DIOS or FLOOPS-ISE™), is the sum of the chemical concentrations of dopants. However, if the total concentration is recomputed inside DESSIS, active concentrations are used.

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

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

- `Nnet` (the net doping concentration used in the Poisson equation, $N_{D^+} - N_{A^-}$, see Section 4.2.1 on page 15.128)
- `Nd` and `Na` (the donor and acceptor concentrations that are used in the main physical models, such as the carrier mobility and lifetime)
- `Ntot` (the total doping concentration $N_D + N_A$ that is used only in surface recombination and incomplete ionization models)

After loading the input doping file, DESSIS has the arrays `DopingConcentration` and `TotalConcentration`, and a number of `SpeciesConcentration` arrays, and uses the following scheme to compute the required simulation doping arrays (`Nnet`, `Nd`, `Na`, `Ntot`):

1. The input doping file does not have any species: `Nnet = DopingConcentration`, `Ntot = TotalConcentration`, or `Ntot = Abs(DopingConcentration)` without `TotalConcentration`; `Nd = 0.5 (TotalConcentration + DopingConcentration)`; `Na = 0.5 (TotalConcentration - DopingConcentration)`.
2. The input doping file has species: `Nd` and `Na` are computed as a sum of donor and acceptor `SpeciesConcentration`; `Nnet = DopingConcentration` or `Nnet = Nd - Na` without `DopingConcentration`; `Ntot = TotalConcentration` or `Ntot = Nd + Na` without `TotalConcentration`.
3. The DESSIS input file contains trap levels (see Chapter 10 on page 15.225) in the `Physics` section with the keyword `Add2TotalDoping` (see Section 10.5 on page 15.229). In this case, the donor trap concentration (from the DESSIS input file) is added to `Nd` and `Ntot`, and the acceptor trap concentration is added to `Na` and `Ntot`.

NOTE Only `Nnet` is recomputed in DESSIS simulation if the incomplete ionization model is activated (see Chapter 6 on page 15.161).

2.14.1 User-defined materials

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

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

Definitions in later files replace or add to the definitions in earlier files. In this way, the local file only needs to contain new materials that the user wants to add.

NOTE The `DATEX` environment variable is no longer used.

To add a new material, add its description to the `Materials` section of `datexcodes.txt`:

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

The `label` value is used as a legend in visualization tools such as MDRAW and Tecplot-ISE.

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

```
Conductor
Insulator
Semiconductor
```

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

```
color = #rrggbb
```

where `rr`, `gg`, and `bb` denote hexadecimal numbers representing the intensity of red, green, and blue, respectively. The values of `rr`, `gg`, and `bb` must be in the range `00` to `ff`. Table 15.41 lists sample values for `color`.

Table 15.41 Sample color values

Color code	Color	Color code	Color
`#000000`	Black	`#ffffff`	White
`#ff0000`	Red	`#40e0d0`	Turquoise
`#00ff00`	Green	`#7fff00`	Chartreuse

Table 15.41 Sample color values

Color code	Color	Color code	Color
`#0000ff`	Blue	`#b03060`	Maroon
`#ffff00`	Yellow	`#ff7f50`	Coral
`#ff00ff`	Magenta	`#da70d6`	Orchid
`#00ffff`	Cyan	`#e6e6fa`	Lavender

Section 18.3 on page 15.321 discusses how compound semiconductors can be defined in DESSIS.

2.14.2 User-defined species

DESSIS supports the most important dopants used in silicon technology: the donors As, P, Sb, N, and the acceptors B and In. For the simulation of other semiconductors such as III–V compounds, the actual dopants (such as Si and Be) are supported only through user-defined species. All such species used during a simulation must be declared in the `Variables` section of the file `datexcodes.txt` (see Section 2.14 on page 15.95). If the incomplete ionization model is activated (see Chapter 6 on page 15.161), the model parameters of user-defined species must be specified in the `Ionization` section of the material parameter file.

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

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

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

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

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

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

```
Plot {
   UserSpeciesConcentration
   UserSpeciesIncompleteConcentration
}
```

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

CHAPTER 3 Mixed-mode DESSIS

3.1 Overview

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

For a multidevice simulation, the command file must include specifications of the mesh (`File` section), contacts (`Electrode` section), and physical models (`Physics` section) for each device. A circuit netlist must be defined to connect the devices (see Figure 15.22), and solve commands must be specified that solve the whole system of devices.

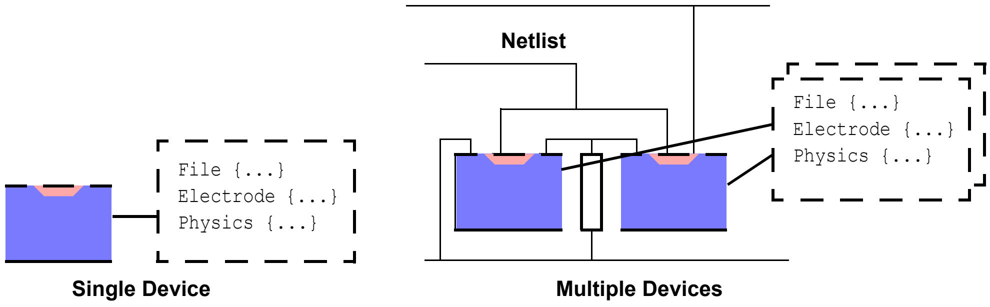


Figure 15.22 Each device in a multidevice simulation is connected with a circuit netlist

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

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

3.1.1 Compact models

DESSIS provides three different types of model:

SPICE	These include compact models from Berkeley SPICE 3 Version 3F5. The BSIM3v3.2, BSIM4.1.0, and BSIMPDv2.2.2 MOS models are also available.
Built-in	There are several special-purpose models.
User-defined	A compact model interface (CMI) is available for user-defined models. These models are implemented in C++ and linked to DESSIS at run-time. No access to the DESSIS source code is necessary.

This section describes the incorporation of compact models in mixed-mode simulations. The Compact Models manual provides references for the three model types.

3.1.2 Hierarchical description of compact models

In DESSIS, the compact models comprise three levels:

Device	This describes the basic properties of a compact model and includes the names of the model, electrodes, thermodes, and internal variables; and the names and types of the internal states and parameters. The devices are predefined for SPICE models and built-in models. For user-defined models, the user must specify the devices.
Parameter set	Each parameter set is derived from a device. It defines default values for the parameters of a compact model. Usually, a parameter set defines parameters that are shared among several instances. Most SPICE and built-in models provide a default parameter set, which can be directly referenced in a circuit description. For more complicated models, such as MOSFETs, the user can introduce new parameter sets.
Instance	Instances correspond to the elements in the DESSIS circuit. Each instance is derived from a parameter set. If necessary, an instance can override the values of its parameters.

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

The parameter sets and instances in a circuit simulation are specified in external SPICE circuit files (see Section 3.2 on page 15.105). These files are recognized by the extension `.scf` and are parsed by DESSIS at the beginning of a simulation.

Table 15.42 lists the built-in models and Table 15.43 presents an overview of the SPICE models.

Table 15.42 Built-in models in DESSIS

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

Table 15.43 SPICE models in DESSIS

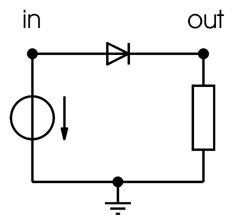
Model	**Device**	**Default parameter set**
Resistor	`Resistor`	`Resistor_pset`
Capacitor	`Capacitor`	`Capacitor_pset`
Inductor	`Inductor`	`Inductor_pset`
Coupled inductors	`mutual`	`mutual_pset`
Voltage-controlled switch	`Switch`	`Switch_pset`
Current-controlled switch	`CSwitch`	`CSwitch_pset`
Voltage source	`Vsource`	`Vsource_pset`
Current source	`Isource`	`Isource_pset`
Voltage-controlled current source	`VCCS`	`VCCS_pset`

Table 15.43 SPICE models in DESSIS

Model	Device	Default parameter set
Voltage-controlled voltage source	`VCVS`	`VCVS_pset`
Current-controlled current source	`CCCS`	`CCCS_pset`
Current-controlled voltage source	`CCVS`	`CCVS_pset`
Junction diode	`Diode`	`Diode_pset`
Bipolar junction transistor	`BJT`	`BJT_pset`
Junction field effect transistor	`JFET`	`JFET_pset`
MOSFET	`Mos1`	`Mos1_pset`
	`Mos2`	`Mos2_pset`
	`Mos3`	`Mos3_pset`
	`Mos6`	`Mos6_pset`
	`BSIM1`	`BSIM1_pset`
	`BSIM2`	`BSIM2_pset`
	`BSIM3`	`BSIM3_pset`
	`BSIM4`	`BSIM4_pset`
	`B3SOI`	`B3SOI_pset`
GaAs MESFET	`MES`	`MES_pset`

3.1.3 Example: Compact models

Consider the following simple rectifier circuit:



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

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

INSTANCE v
```

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

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

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

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

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

The Compact Models manual further explains the SPICE parameters in this example. The DESSIS input file for this simulation can be:

```
File {
   SPICEPath = ". lib spice/lib"
}
System {
   Plot "rectifier" (time() v(in) v(out) i(r 0))
}
Solve {
   Circuit
   NewCurrentPrefix = "transient_"
   Transient (InitialTime = 0 FinalTime = 0.2e-5
         InitialStep = 1e-7 MaxStep = 1e-7) {Circuit}
}
```

The `SPICEPath` in the `File` section is assigned a list of directories. All directories are scanned for `.scf` files (SPICE circuit files).

Check the DESSIS log file to see which circuit files were found and used in the simulation.

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

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

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

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

3.2 SPICE circuit files

Compact models can be specified in an external SPICE circuit file, which is recognized by the extension `.scf`. The declaration of a parameter set can be:

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

This declaration introduces the parameter set `pset` that is derived from the device `dev`. It assigns default values for the given parameters. The device `dev` should have already declared the parameter names. Furthermore, the values assigned to the parameters must be of the appropriate type. Table 15.44 lists the possible parameter types.

Table 15.44 Parameters in SPICE circuit files

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

Similarly, the circuit instances can be declared as:

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

According to this declaration, the instance `inst` is derived from the parameter set `pset`. Its electrodes are connected to the circuit nodes `e0`, `e1`, ..., and its thermodes are connected to `t0`, `t1`, ...

This instance also defines or overrides parameter values. The complete syntax of the input language for SPICE circuit files is given in Compact Models, Section 3.11 on page 16.112.

NOTE The tool SPICE2DESSIS is available to convert Berkeley SPICE circuit files (extension `.cir`) to DESSIS circuit files (extension `.scf`) (see Utilities, Chapter 9 on page 6.65).

3.3 Dessis section

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

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

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

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

3.4 System section

The `System` section defines the netlist of physical devices and circuit elements to be solved. The netlist is connected through circuit nodes. By default, a circuit node is electrical, but it can be declared to be electrical or thermal:

```
Electrical { enode0 enode1 ... }
Thermal { tnode0 tnode1 ... }
```

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

Compact models can be defined in SPICE circuit files (see Section 3.2 on page 15.105). However, instances of compact models can also appear directly in the `System` section of the DESSIS input file:

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

The order of the nodes in the connectivity list corresponds to the electrodes and thermodes in the SPICE device definition (see the Compact Models manual).

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

Physical devices are defined as:

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

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

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

Both diodes have their anode connected to node `1` and their cathode connected to node `2`. In the case of the circuit diode, the device specification defines the order of the electrodes (see Compact Models, Section 3.11 on page 16.112). Conversely, the connectivity for the physical diode must be given explicitly. The names `anode` and `cathode` of the contacts are defined in the grid file of the device. The device types of the physical devices must be defined elsewhere in the input file (with `Dessis` sections) or an external `.dessis` file.

The `System` section can contain the initial conditions of the nodes. Three types of initialization can be specified:

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

In addition, the `Unset` command is available to free a node after a `Set` command.

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

3.4.1 Physical devices

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

```
device_type instance-name (<connectivity list>) {
   <optional device parameters>
}
```

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

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

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

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

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

3.4.2 Circuit devices

SPICE instances can be declared in SPICE circuit files as discussed in Section 3.2 on page 15.105. They can also appear directly in the `System` section of the DESSIS input file, for example:

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

This declaration in the DESSIS input file provides the same information as the equivalent declaration in the SPICE circuit file.

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

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

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

3.4.3 Electrical and thermal netlist

DESSIS allows both electrical and thermal netlists to coexist in the same system, for example:

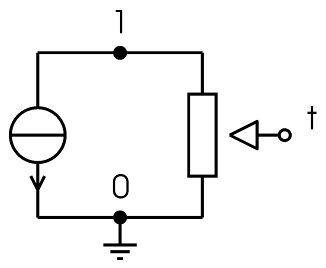
```
System {
   Thermal (ta tc t300)
   Set (t300 = 300)
   Hint (ta = 300 tc = 300)

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

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

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

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

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

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



$$r_0 = \text{rsh} \cdot \frac{l - \text{narrow}}{w - \text{narrow}}$$
$$r(\text{temp}) = r_0 \cdot (1 + \text{tc}_1 \cdot (\text{temp} - \text{tnom}) + \text{tc}_2 \cdot (\text{temp} - \text{tnom})^2) \qquad (15.17)$$

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

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

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

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

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

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

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

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

DESSIS temperatures are defined in kelvin. SPICE temperatures are measured in degree Celsius. Therefore, an offset of –273.15 must be used to convert kelvin to degree Celsius.

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

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

$$\text{Spice temperature} = \text{offset} + c_1 u + c_2 u^2 + c_3 u^3 \tag{15.18}$$

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

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

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

3.4.4 Set, Unset, Initialize, and Hint

The keywords `Set`, `Initialize`, and `Hint` are used to set nodes to a specific value.

`Set` establishes the node value. This value remains fixed during all subsequent simulations until a `Set` or `Unset` command is used in the `Solve` section (see Section 3.8.6 on page 15.121), or the node becomes a `Goal` of a `Quasistationary`, which controls the node itself (see Section 3.8.2 on page 15.116). For a set node, the corresponding equation (that is, the current balance equation for electrical nodes and the heat balance equation for thermal nodes) is not solved, unlike an unset node.

NOTE The `Set and Unset` commands exist in the `Solve` section. These act like the `System` level `Set` but allow more flexibility (see Section 3.4 on page 15.106).

The `Set` command can also be used to change the value of a parameter in a compact model. For example, the resistance of the resistor `r1` changes to 1000 Ω:

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

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

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

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

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

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

For example:

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

3.4.5 System Plot

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

```
Plot (<plot command list>)
Plot <filename> (<plot command list>)
```

where `<plot command list>` is a list of nodes or plot commands as defined in Table 15.45.

NOTE `Plot` commands are case sensitive.

Table 15.45 Parameters for System Plot command

Parameter	Description
`node`	Prints the voltage at the node.
`v(node)`	Prints the voltage at the node.
`v(node1 node2)`	Prints the voltage difference between two given nodes.
`t(node)`	Prints the temperature at the node.
`t(node1 node2)`	Prints the temperature difference between two given nodes.
`i(device node)`	Prints the current that exits in the given device through the given node.
`h(device node)`	Prints the heat that exits in the given device through the given node.
`p(device attribute)`	Prints a given attribute of a given circuit element.
`time()`	Prints the current time.
`freq()`	Prints the current AC analysis frequency.

Two examples are:

```
Plot (a b i(r1 a) p(r1 rT))
Plot "plotfile" (time() v(a b) i(d1 a))
```

NOTE The `Plot` command does not print the time by default. When plotting a transient simulation, the `time()` command must be added.

3.4.6 AC System Plot

An `ACPlot` statement in the `System` section can be used to modify the output in the DESSIS AC plot file:

```
System {
   ACPlot (<plot command list>)
}
```

The `<plot command list>` is the same as the system plot command discussed in Section 3.4.5 on page 15.111.

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

3.5 File section

In the `File` section, one of the following can be specified:

- Output file name and the small signal AC extraction file name
- DESSIS directory path
- Search path for SPICE models and compact models
- Default file names for the devices

Device-specific keywords are defined under the file entry in the `Device` section (see Section 2.2 on page 15.38).

The variables in Table 15.46 can be assigned a file name. Only a file name without an extension is required. DESSIS automatically appends a predefined extension.

Table 15.46 Keyword options in mixed-mode File section

Option	Description
`ACExtract`	Specifies the file in which the results of the small signal AC analysis are stored (`_ac_des.plt`).
`Output`	Logs the screen output (`.log`).

The variables in Table 15.47 represent search paths. They must be assigned a list of directories for which DESSIS searches.

Table 15.47 Search path options in mixed-mode File section

Variable	Description

`DessisPath`	Instructs DESSIS to load all files with the extension `.dessis` in the given directory path. A list of defined physical device types is created internally. The corresponding definition is used if instantiated by the user in the `System` section (and not overwritten by a definition with the same name as the input file) (see Section 3.4 on page 15.106). The directory path has the format `dir1:dir2:dir3`, for example:
	`File {`
	`    DessisPath = ".../devices:/usr/local/tcad/dessis:."`
	`}`

`SPICEPath`	Defines a search path for SPICE circuit files (extension `.scf`). The files are parsed and added to the `System` section of the input file. If the environment variable `ISEROOT_LIB` is defined, the directory `$ISEROOT_LIB/dessis/spice` is automatically added to `SPICEPath`, for example:
	`File {`
	`   SPICEPath = ". lib lib/spice"`
	`}`

`CMIPath`	Defines a search path for compact circuit files (extension `.ccf`) and the corresponding shared object files (extension `.so.arch`). The files are parsed and added to the `System` section of the input file. If the environment variable `ISEROOT_ARCH_OS_LIB` is defined, the directory `$ISEROOT_ARCH_OS_LIB/dessis` is automatically added to `CMIPath`, for example:
	`File {`
	`   CMIPath = ". libcmi"`
	`}`

The default device files are given in the `Device` section. By default, the textual output of the simulator is written to the output file, for example:

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

3.6 SPICE circuit models

DESSIS supports SPICE circuit models for mixed-mode simulations. These models are based on Berkeley SPICE 3 Version 3F5. A detailed description of the Spice models can be found in the Compact Models manual.

3.7 User-defined circuit models

DESSIS provides a compact model interface (CMI) for user-defined circuit models. The models are implemented in C++ by the user and linked to DESSIS at run-time. Access to the DESSIS source code is not required.

To implement a new user-defined model:

1. Provide a corresponding equation for each variable in the compact model. For electrode voltages, compute the current flowing from the device into the electrode. For an internal model variable, use a model equation.
2. Write a formal description of the new compact model. This compact circuit file is read by DESSIS before the model is loaded.
3. Implement a set of interface subroutines C++. DESSIS provides a run-time environment.
4. Compile the model code into a shared object file, which is linked at run-time to DESSIS. A `cmi` script executes this compilation.
5. Use the variable `CMIPath` in the `File` section of the DESSIS command file to define a search path.
6. Reference user-defined compact models in compact circuit files (with the extension `.ccf`) or directly in the `System` section of the DESSIS input file.

3.8 Solve section

The `Solve` commands do not fundamentally change when used in mixed mode. The major differences are with the `Coupled` command, the extra goals available for the `Quasistationary` command, and the `ACCoupled` and `Continuation` commands that only work in mixed mode.

3.8.1 Coupled command

The `Coupled` command in mixed mode addresses contact and circuit equation variables. Devices can be selected individually or together.

3.8.1.1 Circuit and contact equation-variable keywords

The `Coupled` command takes a set of equation-variable pairs as parameters. The `Contact` and `Circuit`, and `TContact` and `TCircuit` equation-variable pairs are introduced for mixed-mode problems. Table 15.48 provides the full list of equation-variable pair keywords. These four keywords are accessible if the keyword `NoAutomaticCircuitContact` is specified in the `Math` section (at the top level).

Table 15.48 Keywords of equation-variable pairs for Coupled command

Keyword	Corresponding equation	Corresponding variable
`Poisson`	Poisson	Electrostatic potential
`Contact`	Electrical contact interface	Electrostatic potential
`TContact`	Thermal contact interface	Temperature
`Circuit`	Electrical circuit	Voltage, current
`TCircuit`	Thermal circuit	Temperature
`Electron`	Electron continuity	Electron density
`Hole`	Hole continuity	Hole density

Table 15.48 Keywords of equation-variable pairs for Coupled command

Keyword	Corresponding equation	Corresponding variable
`Temperature`	Temperature	Temperature
`eTemperature`	Electron temperature	Electron temperature
`hTemperature`	Hole temperature	Hole temperature

The keyword `Circuit` controls the resolution of the electrical circuit models and nodes. `Contact` controls the resolution of the electrical interface condition at the contacts. Similarly, `TContact` and `TCircuit` control the resolution of the thermal interface condition and thermal circuit, respectively. By default, the keywords `TContact` and `TCircuit` are not used because the Poisson equation-variable also covers the contact and circuit domains.

When the keyword `NoAutomaticCircuitContact` is used in the `Math` statement, the `Poisson` keyword only covers the device. The `TCircuit` and `TContact` keywords can then be used for the circuit and contact parts (see Figure 15.23).

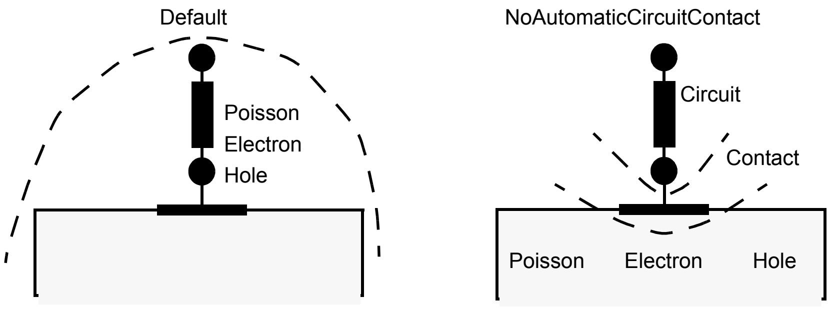


Figure 15.23 Range of equation-variable keywords Circuit, Contact, Poisson, Electron, and Hole

3.8.1.2 Selecting individual devices

The default usage of an equation–variable keyword such as `Poisson` activates the given equations and variables for all devices. With complex multiple-device systems, such an action is not always desirable especially when a fully consistent solution has not yet been found. DESSIS allows each equation–variable pair to be restricted to a specific device by adding the name of the device instance to the equation–variable keyword separated with a period. The syntax is:

```
<identifier>.<equation-variable>
```

Device-specific solutions are used to obtain the initial solution for the whole system. For example, with two or more devices, it is often better to solve each device individually before coupling them all. Such a scheme can be written as:

```
System {
  ... device1 ...
  ... device2 ...
}
Solve {
  # Solve Circuit equation-variable
  Circuit

  # Solve poisson and full coupled for each device
  Coupled { device1.Poisson device1.Contact}
```

```
    Coupled { device1.Poisson device1.Contact
              device1.Electron device1.Hole }
    Coupled { device2.Poisson device2.Contact }
    Coupled { device2.Poisson device2.Contact
              device2.Electron device2.Hole }

    # Solve full coupled over all devices
    Coupled { Circuit Poisson Contact Electron Hole }
  }
```

3.8.2 Quasistationary command

The `Quasistationary` command is extended in mixed mode to include goals on nodes and circuit model parameters. Table 15.49 defines the syntax of these goals.

Table 15.49 Mixed-mode parameters for Quasistationary command

Keyword	Description
`Goal { Node=<string> Voltage=<float> }`	Sets the new target voltage to a specified node (name).
`Goal { Parameter = <i-name>.<p-name> value = <float> }`	Sets the new target value to a specified parameter (`p-name`) of a device instance (`i-name`).

The goal on a node is usually used on a node that has been fixed with the `Set` or `Initialize` command in the `System` section. For example, the node `a` is set to 1 V and ramped to 10 V:

```
System {
  Resistor_pset r1(a 0){resistance = 1}
  Set(a=1)
}

Solve{
  Circuit
  Quasistationary( Goal{Node=a Voltage=10} ){ Circuit }
}
```

A goal on circuit model parameters can be used to change the configuration of a system. Any circuit model parameter can be changed. For example, the resistor `r1` is ramped from $1\ \Omega$ to $0.1\ \Omega$:

```
System {
  Resistor_pset r1(a 0){resistance = 1}
  Set(a=1)
}

Solve{
  Circuit
  Quasistationary(Goal{Parameter=r1."resistance" Value=0.1})
    { Circuit }
}
```

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

3.8.3 ACCoupled: Small-signal AC analysis

An `ACCoupled` solve section is an extension of a `Coupled` section with an extra set of parameters allowing small-signal AC analysis. Table 15.50 describes these parameters.

Table 15.50 Parameters for Small Signal AC Analysis

Parameter	Description
`StartFrequency = <float>`	Analyzes lower frequency.
`EndFrequency = <float>`	Analyzes upper frequency.
`NumberOfPoints = <integer>`	Defines the number of frequencies with which to perform the analysis.
`Linear`	Applies linear intervals between the frequencies.
`Decade`	Applies logarithmic intervals between the frequencies.
`Node(<node-name> <node-name>...)`	Defines nodes over which the small-signal analysis is performed.
`Exclude(<device-name> ...<device-name>...)`	Defines devices that must not be included in the small-signal analysis.
`ObservationNode(<node-name> ...<node-name>...)`	Activates noise analysis (see Chapter 15 on page 15.291).
`Optical`	Switches on optical AC analysis (see Section 3.8.4 on page 15.119).
`ACCompute (<options>)`	Perform an AC or noise analysis only for selected bias points inside a `Quasistationary` command.
`ACPlot=<file name prefix>`	Activates plotting of the responses of the solution variables to the AC signals. The string specified is used as a prefix for the names of the files to which the responses are written.
`NoisePlot=<file name prefix>`	Specifies a prefix for a file name (see Chapter 15 on page 15.291).
`ACExtract=<file name prefix>`	Specifies a prefix for the name of the file to which the results of AC analysis are written (overrides the specification from the `File` section).

The options `StartFrequency`, `EndFrequency`, `NumberOfPoints`, `Linear`, and `Decade` are used to select the frequencies at which the analysis is performed and the frequency distribution.

A `Node` list must be given. With it, for each frequency, the compact equivalent small signal model is generated between the given nodes. This conductive-capacitive matrix is stored in an `ACExtract` file specified in the `File` or `ACCoupled` statement (default is `extraction_ac_des.plt`). This ASCII file contains the frequency, voltages at the nodes, and entries of the matrix. If the file name prefix `ACPlot` is specified, for each node in the `Node` list and for each frequency, DESSIS writes a file with the (complex-valued) derivatives of the solution variables with respect to voltage at the node.

The `Exclude` list is used to remove a set of circuit or physical devices from the AC analysis. Typically, the power supply is removed so as not to short-circuit the AC analysis, but the list can also be used to isolate a single device from a whole circuit.

NOTE The system analyzed consists of the equations specified in the body of the `ACCoupled` statement, without the instances removed by the `Exclude` list. The `Exclude` list only specifies instances, therefore, all equations of these instances are removed.

The `ACCompute` option controls AC or noise analysis performances within a `Quasistationary` command. The parameters in `ACCompute` are identical to the parameters in the `Plot` and `Save` commands (see Table 15.28 on page 15.74), for example:

```
Quasistationary (...) {
   ACCoupled (...
      ACCompute (Time = (0; 0.01; 0.02; 0.03; 0.04; 0.05)
               Time = (Range = (0.9 1.0) Intervals = 4)
               Time = (Range = (0.1 0.2); Range = (0.7 0.8))
               When (Node = in Voltage = 1.5))
      {...}
}
```

In this example, an AC analysis is performed only for the time points:

```
t = 0, 0.01, 0.02, 0.03, 0.04, 0.05
t = 0.9, 0.925, 0.95, 0.975, 1.0
```

and for all time points in the intervals [0.1, 0.2] and [0.7, 0.8]. Additionally, an AC analysis is triggered whenever the voltage at the node `in` reaches the 1.5 V threshold.

If the AC or noise analysis is suppressed by the `ACCompute` option, an `ACCoupled` command behaves like an ordinary `Coupled` command inside a `Quasistationary`.

3.8.3.1 Example: AC analysis of simple device

This example illustrates AC analysis of a simple device. A 1D resistor is connected to ground (through resistor `to_ground`) and to a voltage source `drive` at reverse bias of –3 V. After calculating the initial voltage point at –3 V, the left voltage is ramped to 1 V in 0.1 V increments. The AC parameters between nodes `left` and `right` are calculated at one frequency $f = 10^3$ Hz. The circuit element `drive` and `to_ground` are excluded from the AC calculation.

By including circuits, complete Bode plots can be performed:

```
Dessis resist {
   Electrode {{Name=anode Voltage=-3 resist=1}
         {Name=cathode Voltage=0 resist=1}}

   File {Grid   = "resist"
         Doping = "resist"}

   Physics {Mobility (DopingDep)
         Recombination (SRH(DopingDep) Auger)}
}

System {
   resist 1d (anode=left cathode=right)
         Vsource_pset drive(left right){dc = -3}
         Resistor_pset to_ground (right 0){resistance=0}
}
```

```
Math {method=blocked NoAutomaticCircuitContact}

Solve {
   Circuit
         Plugin (Digits=2) {Poisson Electron Hole}
         Coupled {poisson electron hole contact circuit}
         ACCoupled (StartFrequency=1e3 EndFrequency=1e6
               NumberOfPoints=4 Decade
               Iterations=0
               Node(left right)
               Exclude(drive to_ground))
               {poisson electron hole contact circuit}
         Quasistat (MaxStep=0.025 InitialStep=0.025
            Goal{Parameter=drive.dc Value=1}) {
            ACCoupled(StartFrequency=1e3 EndFrequency=1e6
               NumberOfPoints=4 Decade
               Node(left right)
               Exclude(drive to_ground))
               {electron hole poisson Circuit Contact}
         }
}
```

3.8.4 Optical AC analysis

Optical AC analysis calculates the quantum efficiency as a function of the frequency of the optical signal. This technique is based on the AC analysis technique, and provides real and imaginary parts of the quantum efficiency versus the frequency.

To start optical AC analysis, add the keyword `Optical` in an `ACCoupled` statement, for example:

```
ACCoupled ( StartFrequency=1.e4 EndFrequency=1.e9
            NumberOfPoints=31 Decade Node(a c)
            Optical Exclude(v1 v2) )
            { poisson electron hole }
```

3.8.5 Continuation: An alternative ramping method

The simulation of an I(V) curve can be run by using the keyword `Continuation` in the `Solve` section. The parameters in braces are the same type as for `Transient` and `Quasistationary`. Some control parameters must be given in parentheses in the same way as for the `Plugin` statement, for example:

```
Continuation (<Control Parameters>) {
   coupled (iterations=5) { poisson electron hole }
}
```

Table 15.51 on page 15.120 summarizes the control parameters of the `Continuation` command. To allow general usage in a mixed-mode environment, `Continuation` requires the use of a system level node. Only one node, defined in the connectivity list of a device, is allowed for this method of solution. The node can be set or not set (with the `Set` or `Initialize` command). If the node is not set, it continues to float after the end of the continuation.

Table 15.51 Parameters for continuation method

Control parameter	Description
`Node=<node-name>`	Specifies which node to ramp. Node must be set; otherwise, it is set during the continuation and unset afterward.
`InitialVstep=<float>`	Initial voltage step.
`MinVoltage=<float>`	Lower voltage limit.
`MaxVoltage=<float>`	Upper voltage limit.
`MinCurrent=<float>`	Lower current limit.
`MaxCurrent=<float>`	Upper current limit.
`MinStep=<float>`	Minimum step for the internal arc length variable (default 1×10^{-5}).
`MaxStep=<float>`	Maximum step for the internal arc length variable.
`Increment=<float>`	Factor used to augment arc length on successful solve (default 2).
`Decrement=<float>`	Factor used to diminish arc length on failure to solve (default 2).
`Error=<float>`	Absolute error used to control continuation (default 5×10^{-2}).
`Digits=<float>`	Relative error used to control continuation (default 1×10^{-3}).
`LogCurrent`	Treats current on a log scale. This can improve convergence when the current changes by many orders of magnitude.

The first step of the continuation is always a voltage-controlled step. For this, the user must supply an initial voltage step in the control parameter list using the `InitialVstep` statement. The continuation proceeds automatically, until the values given by any of the `MinVoltage`, `MaxVoltage`, `MinCurrent`, or `MaxCurrent` parameters are exceeded.

NOTE Be conservative when defining these parameters because they are also used for the scaling of the continuation equation.

In addition, the parameters `MaxStep`, `Increment`, `Decrement`, `Error`, and `Digits` can be specified. Their definitions are the same as for the `Transient` and `Quasistationary` statements, except that they measure the scaled arc length.

If the current extends over several orders of magnitude, it can be useful to state an additional parameter `LogCurrent`. If this flag is set, the curve log(I(V)) is traced instead of I(V). The resulting graph is expected to be identical, but it can be useful for the convergence behavior.

NOTE As a general rule, use `LogCurrent` when the I(V) curve is to be viewed on a logarithmic graph. If `LogCurrent` is used, be careful that the current does not become negative during the simulation.

3.8.5.1 Example: Solve entry for continuation

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

```
Solve{ ...
   Continuation ( node=node1 InitialVstep=-0.003
                  MaxVoltage=3 MaxCurrent=0
                  MinVoltage=-3 MinCurrent=-5e-3 ) {
   Coupled (iterations=5) { poisson electron hole }
   }
}
```

This specifies that the I(V) curve must be traced at the circuit node `node1`. The initial voltage-controlled step is –0.003 V, the voltage range is –3 V to 3 V, and the current range is –5 mA to 0 A.

3.8.6 Set and Unset section

The keyword `Set` in the `Solve` section is used to set node values during a part of the simulation. The `Set` command in the `Solve` section is position dependent. This allows a node to be set to a previously computed value. The `Set` command takes a list of nodes with optional values as parameters. If a value is given, the node is set to that value. If no value is given, the node is set to its current value. The nodes are set until the end of the DESSIS run or the next `Unset` command with the specified node.

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

The `Set` command can also be used to affect the boundary condition type of electrodes in a single device mode (see Section 2.9.10 on page 15.73).

3.9 Math section

The keyword `AutomaticCircuitContact` (active by default) controls the automatic inclusion of the circuit and contact equations in a mixed-mode simulation. If only the Poisson equation is solved, no additional equations are added. However, if additional equations to `Poisson` appear in a `Coupled` statement, the circuit and contact equations are also added.

Therefore:

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

is equivalent to:

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

NOTE `AutomaticCircuitContact` does not add the circuit and contact equations if `Poisson` is restricted to instances, for example:

```
Coupled { device1.Poisson device1.Electron device1.Hole
          device2.Poisson device2.Electron device2.Hole }
```

If the keyword `NoAutomaticCircuitContact` appears in the `Math` section, DESSIS does not add the circuit and contact equations automatically (see Section 3.8.1.1 on page 15.114). The following SPICE circuit parameters can be specified in the global `Math` section:

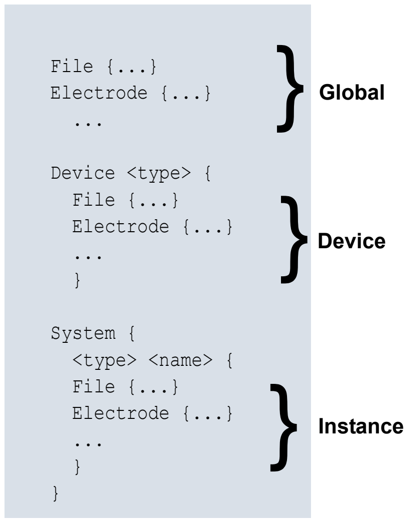
```
Spice_Temperature = ...
Spice_gmin = ...
```

The value of `Spice_Temperature` denotes the global SPICE circuit temperature. Its default value is 300.15 K. The parameter `Spice_gmin` refers to the minimum conductance in SPICE. The default value is 10^{-12} S .

3.10 Using mixed-mode simulation

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

3.10.1 From single device file to multidevice file



```
File {...}
Electrode {...}
  ...

Device <type> {
  File {...}
  Electrode {...}
  ...
  }

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

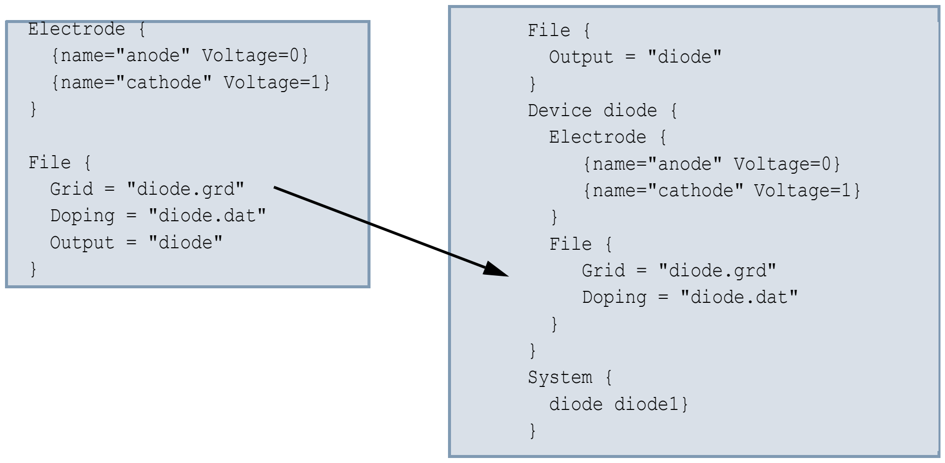
Figure 15.24 Three levels of device definition

The DESSIS command file accepts both single-device and multidevice problems. Although the two forms of input look different, they fit in the same input syntax pattern. This is possible because the input file has multiple levels of definitions and there is a built-in default mechanism for the `System` section. The complete input syntax allows for three levels of device definition: global, device, and instance (see Figure 15.24). The three levels are linked in that the global level is the default for the device level and instance level.

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

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

This translation process can be performed manually by creating a `Device` and `System` section with a single entry (see Figure 15.25).

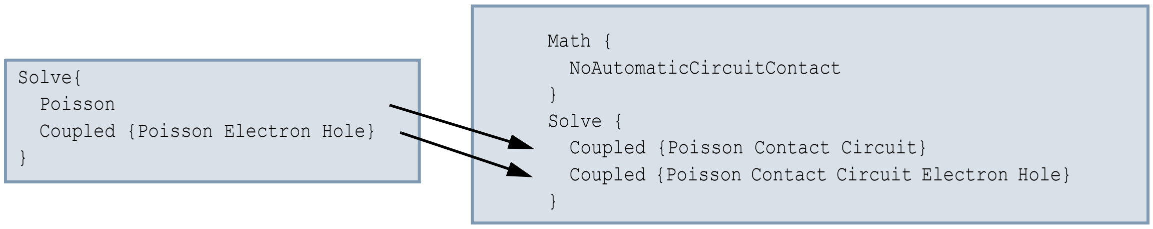


```
Electrode {
  {name="anode" Voltage=0}
  {name="cathode" Voltage=1}
}

File {
  Grid = "diode.grd"
  Doping = "diode.dat"
  Output = "diode"
}
```

```
File {
  Output = "diode"
}
Device diode {
  Electrode {
     {name="anode" Voltage=0}
     {name="cathode" Voltage=1}
  }
  File {
     Grid = "diode.grd"
     Doping = "diode.dat"
  }
}
System {
  diode diode1}
}
```

Figure 15.25   Translating a single device syntax to mixed-mode form

The `Solve` section can also be converted if the flag `NoAutomaticCircuitContact` is used (see Figure 15.26).



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

```
Math {
  NoAutomaticCircuitContact
}
Solve {
  Coupled {Poisson Contact Circuit}
  Coupled {Poisson Contact Circuit Electron Hole}
}
```

Figure 15.26   Translating a default solve syntax to a NoAutomaticCircuitContact form

In this case, all occurrences of the keyword `Poisson` must be expanded to the three keywords `Poisson Contact Circuit`.

3.10.2 File-naming convention: Mixed-mode extension

A `File` section can be defined at all levels of an input command file. Therefore, a default file name can be potentially included by more than one device, for example:

```
Device res {
  File { Save="res" ... }
  ...
}
System {
  res r1
  res r2
}
```

Both `r1` and `r2` use the save device parameters set up in the definition of `res`. Therefore, they have in principle the same default for `Save` (that is, `res`). Since it is impractical to save both devices under the same name, the names of the device instances (that is, `r1` and `r2`) are concatenated to the file name to produce the files `res.r1` and `res.r2`. This process of file name extension is performed for the file parameters `Save`, `Current`, `Path`, and `Plot`, and is also performed when the file name is copied from the global default to a device type declaration.

In practice, three possibilities exist:

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

Table 15.52 summarizes these possibilities.

Table 15.52 File modification for Save, Current, Path, and Plot commands

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

Part II Physics in DESSIS

This part of the DESSIS manual contains the following chapters:

CHAPTER 4 Introduction to physics in DESSIS

4.1 Overview

Physical phenomena in semiconductor devices are very complicated and, depending on applications, are described by partial differential equations of different level of complexity. Coefficients and boundary conditions of equations (such as mobility, generation–recombination rate, material-dependent parameters, interface and contact boundary conditions) can be very complicated and can depend on microscopic physics, the structure of the device, and the applied bias.

DESSIS allows for arbitrary combinations of transport equations and physical models, which allows for the possibility to simulate all spectrums of semiconductor devices, from power devices to deep submicron devices and sophisticated heterostructures. This chapter describes the formulation of physical models and equations.

4.2 Transport equations

Depending on the device under investigation and the level of modeling accuracy required, the user can select four different simulation modes:

Drift-diffusion	Isothermal simulation, described by basic semiconductor equations. Suitable for low-power density devices with long active regions.
Thermodynamic	Accounts for self-heating. Suitable for devices with low thermal exchange, particularly, high-power density devices with long active regions.
Hydrodynamic	Accounts for energy transport of the carriers. Suitable for devices with small active regions.
Monte Carlo	Allows for full band Monte Carlo device simulation in the selected window of the device.

The equations for the drift-diffusion, thermodynamic, and hydrodynamic modes are presented in this section.

The Monte Carlo mode allows the Boltzmann transport equation to be solved in a selected window of the device. Monte Carlo simulation is provided by the device simulator SPARTA™ (see the SPARTA manual for more information).

The equations are formulated initially under the assumption of Boltzmann statistics for electrons and holes. See Section 4.4 on page 15.137 for a description of the corresponding equations for Fermi–Dirac statistics.

The transport model can be selected independently for either carrier in DESSIS, or transport can be neglected by assuming a constant quasi-Fermi level for a nonselected carrier. The same applies for the energy balance equation. If it is solved for the temperature of one carrier only, the temperature of the other carrier is assumed to be equal to the lattice temperature. The `Solve` statement (see Section 2.9 on page 15.54) specifies the set of equations be solved.

4.2.1 Basic equations for semiconductor device simulation

The three governing equations for charge transport in semiconductor devices are the Poisson equation and the electron and hole continuity equations. The Poisson equation is:

$$\nabla\varepsilon \cdot \nabla\psi = -q\left(p - n + N_{D^+} - N_{A^-}\right) \tag{15.19}$$

where ε is the electrical permittivity, q is the elementary electronic charge, n and p are the electron and hole densities, and N_{D+} is the number of ionized donors, and N_{A-} is the number of ionized acceptors.

The keyword for the Poisson equation is `Poisson`. The keywords for the electron and hole continuity equations are `electron` and `hole`, respectively. For example, in the `Coupled` command, they are written as:

$$\nabla \cdot \vec{J}_n = qR + q\frac{\partial n}{\partial t} \qquad -\nabla \cdot \vec{J}_p = qR + q\frac{\partial p}{\partial t} \tag{15.20}$$

where R is the net electron–hole recombination rate (see Chapter 9 on page 15.201), $\vec{J}_n$ is the electron current density, and $\vec{J}_p$ is the hole current density.

4.2.2 Drift-diffusion model

The drift-diffusion model is widely used for the simulation of carrier transport in semiconductors and is defined by the basic semiconductor equations (see Section 4.2.1), where current densities for electrons and holes are given by:

$$\vec{J}_n = -nq\mu_n\nabla\phi_n \tag{15.21}$$

$$\vec{J}_p = -pq\mu_p\nabla\phi_p \tag{15.22}$$

where μ_n and μ_p are the electron and hole mobilities (see Chapter 8 on page 15.175), and ϕ_n and ϕ_p are the electron and hole quasi-Fermi potentials, respectively (see Section 4.3 on page 15.136).

4.2.3 Thermodynamic model

The thermodynamic (or nonisothermal) model [2] extends the drift-diffusion approach to account for electrothermal effects, under the assumption that the charge carriers are in thermal equilibrium with the lattice. Therefore, the carrier temperatures and the lattice temperature are described by a single temperature T.

4.2.3.1 Model description

The thermodynamic model is defined by the basic set of partial differential equations (Eq. 15.19) and (Eq. 15.20), and the lattice heat flow equation (Eq. 15.25). The relations (Eq. 15.21) and (Eq. 15.22) are generalized to include the temperature gradient as a driving term:

$$\vec{J}_n = -nq\mu_n(\nabla\phi_n + P_n\nabla T) \tag{15.23}$$

$$\vec{J}_p = -pq\mu_p(\nabla\phi_p + P_p\nabla T) \tag{15.24}$$

where P_n and P_p are the absolute thermoelectric powers [11] (see Section 24.5 on page 15.367). For accurate simulation of self-heating effects, this extra driving force for the current can be included by specifying the keyword `Thermodynamic` in the `Physics` section of the input file.

To calculate the temperature distribution in the device due to self-heating, the following equation is solved:

$$\begin{aligned} c\frac{\partial T}{\partial t} - \nabla\cdot\kappa\nabla T = -\nabla\cdot[(P_nT+\phi_n)\vec{J}_n + (P_pT+\phi_p)\vec{J}_p] - \left(E_C + \frac{3}{2}k_BT\right)\nabla\cdot\vec{J}_n \\ -\left(E_V - \frac{3}{2}k_BT\right)\nabla\cdot\vec{J}_p + qR(E_C - E_V + 3k_BT) \end{aligned} \tag{15.25}$$

where κ is the thermal conductivity (see Section 24.3 on page 15.366) and c is the lattice heat capacity (see Section 24.1 on page 15.365). E_C and E_V are the conduction and valence band energies, respectively, and R is the recombination rate.

4.2.3.2 Syntax and implementation

To activate a nonisothermal simulation, the keyword `Temperature` (synonyms are `LatticeTemperature` or `LTemperature`) must be specified inside the `Coupled` command of the `Solve` section (see Section 2.9 on page 15.54). To activate extra terms in the current density equations (due to the gradient of the temperature), the keyword `Thermodynamic` must be specified in the `Physics` section of the input file. Without `Thermodynamic` (and `Temperature` still included in the `Solve` section), DESSIS uses (Eq. 15.26) and (Eq. 15.27) instead of (Eq. 15.23) and (Eq. 15.24), and uses (Eq. 15.30) instead of (Eq. 15.25) (see Section 4.2.4 on page 15.130).

DESSIS allows the total heat generation rate to be plotted and the separate components of the total heat to be estimated and plotted. The total heat generation rate is the term on the right-hand side of (Eq. 15.25). It is plotted using the `TotalHeat` keyword in the `Plot` section of the DESSIS input file. The total heat is calculated only when DESSIS solves the temperature equation.

The formulas to estimate individual heating mechanisms and the appropriate keywords for use in the Section 2.6 on page 15.52 of the DESSIS input file are shown in Table 15.53. The individual heat terms that are used for plotting and that are written to the `.log` file are less accurate than those DESSIS uses to solve the transport equations. They serve as illustrations only.

Table 15.53 Terms and keywords used in Plot section of command file

Heat name	**Keyword**	**Formula**
Electron Joule heat	`eJouleHeat`	$\frac{\lvert\vec{J}_n\rvert^2}{qn\mu_n}$
Hole Joule heat	`hJouleHeat`	$\frac{\lvert\vec{J}_p\rvert^2}{qp\mu_p}$
Recombination heat	`RecombinationHeat`	$qR(\phi_p - \phi_n + T(P_p - P_n))$

Table 15.53 Terms and keywords used in Plot section of command file

Heat name	Keyword	Formula
Thomson plus Peltier heat [156]	`ThomsonHeat`	$-\vec{J}_n \cdot T \cdot \nabla P_n - \vec{J}_p \cdot T \cdot \nabla P_p$
Peltier heat	`PeltierHeat`	$-T\left(\frac{\partial P_n}{\partial n}\vec{J}_n \cdot \nabla n + \frac{\partial P_p}{\partial p}\vec{J}_p \cdot \nabla p\right)$

To plot the lattice heat flux vector $\kappa\nabla T$, specify the keyword `lHeatFlux` in the `Plot` section (see Section 2.6 on page 15.52).

4.2.4 Hydrodynamic model

With continued scaling into the deep submicron regime, neither internal nor external characteristics of state-of-the-art semiconductor devices can be described properly using the conventional drift-diffusion transport model. In particular, the drift-diffusion approach cannot reproduce velocity overshoot and often overestimates the impact ionization generation rates. The Monte Carlo method for the solution of the Boltzmann kinetic equation is the most general approach, but because of its high computational requirements, it cannot be used for the routine simulation of devices in an industrial setting.

In this case, the hydrodynamic (or energy balance) model provides a very good compromise. Since the work of Stratton [12] and Bløtekjær [13], there have been many variations of this model [15]–[29]. The full formulation, including the so-called convective terms [16], consists of eight partial differential equations (PDEs) [3][14][17], while the simpler form (no convective terms) includes six PDEs [18]–[20][28].

4.2.4.1 Syntax and implementation

To perform a hydrodynamic simulation, the keyword `eTemperature` (synonym `ElectronTemperature`) or `hTemperature` (synonym `HoleTemperature`) must be specified inside the `Coupled` command of the `Solve` section (see Section 2.9 on page 15.54).

In addition, to activate the hydrodynamic model, the keyword `Hydrodynamic` (or `Hydro`) must be specified in the `Physics` section. If only one carrier temperature equation is to be solved, `Hydro` must be specified with the appropriate parameter, either `Hydro(eTemp)` or `Hydro(hTemp)`. If the hydrodynamic model is not activated for a particular carrier type, DESSIS merges the temperature for this carrier with the lattice temperature. That is, these temperatures are equal, and their heating terms (see the right-hand sides of (Eq. 15.28), (Eq. 15.29), and (Eq. 15.30)) are added.

By default, DESSIS energy conservation equations do not include generation–recombination heat sources. To activate them, the keyword `RecGenHeat` must be specified in the `Physics` section.

To plot the electron, hole, and lattice heat flux vectors $\vec{S}_n, \vec{S}_p, \vec{S}_L$ (see (Eq. 15.31), (Eq. 15.32), and (Eq. 15.33)), specify the corresponding keywords `eHeatFlux`, `hHeatFlux`, and `lHeatFlux` in the `Plot` section (see Section 2.6).

4.2.4.2 Physical model description

The transport model implemented in DESSIS is based on the approach involving the solution of six PDEs. In the hydrodynamic model, the carrier temperatures T_n and T_p are assumed to not equal the lattice temperature T_L. Together with basic semiconductor equations, up to three additional equations can be solved to find the temperatures. In general, the model consists of the basic set of PDEs (the Poisson equation and continuity equations, see Section 4.2.1 on page 15.128) and the energy conservation equations for electrons, holes, and the lattice.

In the hydrodynamic case, current densities are defined as:

$$\vec{J}_n = q\mu_n\left(n\nabla E_C + k_B T_n \nabla n + f_n^{td} k_B n \nabla T_n - 1.5 n k_B T_n \nabla \ln m_e\right) \tag{15.26}$$

$$\vec{J}_p = q\mu_p\left(p\nabla E_V - k_B T_p \nabla p - f_p^{td} k_B p \nabla T_p - 1.5 p k_B T_p \nabla \ln m_h\right) \tag{15.27}$$

where E_C and E_V are the conduction and valence band energies, respectively. The first term takes into account the contribution due to the spatial variations of electrostatic potential, electron affinity, and the band gap. The three remaining terms in (Eq. 15.26) and (Eq. 15.27) take into account the contribution due to the gradient of concentration, the carrier temperature gradients, and the spatial variation of the effective masses m_e and m_h.

The energy balance equations read:

$$\frac{\partial W_n}{\partial t} + \nabla \cdot \vec{S}_n = \vec{J}_n \cdot \nabla E_C + \left.\frac{dW_n}{dt}\right|_{coll} \tag{15.28}$$

$$\frac{\partial W_p}{\partial t} + \nabla \cdot \vec{S}_p = \vec{J}_p \cdot \nabla E_V + \left.\frac{dW_p}{dt}\right|_{coll} \tag{15.29}$$

$$\frac{\partial W_L}{\partial t} + \nabla \cdot \vec{S}_L = \left.\frac{dW_L}{dt}\right|_{coll} \tag{15.30}$$

where the energy fluxes are:

$$\vec{S}_n = -\frac{5r_n}{2}\left(\frac{k_B T_n}{q}\vec{J}_n + f_n^{hf}\hat{\kappa}_n \nabla T_n\right) \tag{15.31}$$

$$\vec{S}_p = -\frac{5r_p}{2}\left(\frac{-k_B T_p}{q}\vec{J}_p + f_p^{hf}\hat{\kappa}_p \nabla T_p\right) \tag{15.32}$$

$$\vec{S}_L = -\kappa_L \nabla T_L \tag{15.33}$$

$$\hat{\kappa}_n = \frac{k_B^2}{q} n \mu_n T_n \tag{15.34}$$

$$\hat{\kappa}_p = \frac{k_B^2}{q} p \mu_p T_p \tag{15.35}$$

The parameters r_n, r_p, f_n^{td}, f_p^{td}, f_n^{hf}, and f_p^{td} are accessible in the DESSIS parameter file. Different values of these parameters can significantly influence the physical results, such as velocity distribution and possible spurious velocity peaks. By changing these parameters, DESSIS can be tuned to a very wide set of hydrodynamic/energy balance models as described in the literature, from Stratton to Bløtekjær [12][13][24][28][29]. The DESSIS default parameter values are:

$$r_n = r_p = 0.6 \tag{15.36}$$

$$f_n^{td} = f_p^{td} = 0 \tag{15.37}$$

$$f_n^{hf} = f_p^{hf} = 1 \tag{15.38}$$

This model corresponds to the energy balance formulation originated by Stratton [12] and takes into account that the microscopic relaxation time has the form $\tau \sim E^{\nu}$ where E is the mean carrier energy. The default constants were obtained for $\nu = -1$, which means that the carrier mobility is inversely proportional to the carrier temperature.

By changing constants f_n^{hf} and r_n, the convective contribution can be changed to the flux and constant C_n in the Widemann–Franz law:

$$\kappa_n = \left(\frac{5}{2} + C_n\right)\frac{k_B^2}{q} n \mu_n T_n \tag{15.39}$$

because following from (Eq. 15.31), convective contributions and diffusive contributions to the flux are equal to:

$$-\frac{5 r_n}{2} \frac{k_B T_n}{q} \vec{J}_n \quad \text{and} \quad -\frac{5 r_n}{2} f_n^{hf} \hat{\kappa}_n \nabla T_n \tag{15.40}$$

respectively.

With the DESSIS default set of transport parameters, (Eq. 15.39) has the form:

$$\kappa_n = \frac{3}{2} \cdot \frac{k_B^2}{q} n \mu_n T_n \tag{15.41}$$

If the hydrodynamic model is based on the Bløtekjær approach, the parameters should be:

$$f_n^{td} = f_p^{td} = f_n^{hf} = f_p^{hf} = 1\,, \qquad r_n = r_p = 1 \tag{15.42}$$

The collision terms are expressed by this set of equations:

$$\left.\frac{dW_n}{dt}\right|_{\text{coll}} = -H_n - \frac{W_n - W_{n0}}{\tau_{en}} \tag{15.43}$$

$$\left.\frac{dW_p}{dt}\right|_{\text{coll}} = -H_p - \frac{W_p - W_{p0}}{\tau_{ep}} \tag{15.44}$$

$$\left.\frac{dW_L}{dt}\right|_{\text{coll}} = H_L + \frac{W_n - W_{n0}}{\tau_{en}} + \frac{W_p - W_{p0}}{\tau_{ep}} \tag{15.45}$$

Here, H_n, H_p, and H_L are the energy gain/loss terms due to generation–recombination processes. The expressions used for these terms are based on approximations [20] and have the following form for the major generation–recombination phenomena:

$$H_n = 1.5k_BT_n((R^{SRH} + R^{rad}) + R_n^{trap}) + E_g(R_n^A - G_n^{ii}) \tag{15.46}$$

$$H_p = 1.5k_BT_p((R^{SRH} + R^{rad}) + R_p^{trap}) + E_g(R_p^A - G_p^{ii}) \tag{15.47}$$

$$H_L = [R^{SRH} + 0.5(R_n^{trap} + R_p^{trap})](E_g + 1.5k_BT_n + 1.5k_BT_p) \tag{15.48}$$

where:

R^{SRH} is the Shockley–Read–Hall (SRH) recombination rate (see Section 9.1 on page 15.201).

R^{rad} is the radiative recombination rate (see Section 9.6 on page 15.211).

R_n^A and R_p^A are Auger recombination rates (see Section 9.7 on page 15.212) related to electrons and holes.

G_n^{ii} and G_p^{ii} are impact ionization rates (see Section 9.9 on page 15.213).

R_n^{trap} and R_p^{trap} are the recombination rates through trap levels (see Chapter 10 on page 15.225).

Surface recombination is taken into account in a way similar to bulk SRH recombination. Usually, the influence of H_n, H_p, and H_L is small, so they are not activated by default. To take these energy sources into account, the keyword `RecGenHeat` must be specified in the `Physics` section.

The energy densities W_n, W_p, and W_L are given by:

$$W_n = nw_n = n\left(\frac{3k_BT_n}{2}\right) \tag{15.49}$$

$$W_p = pw_p = p\left(\frac{3k_BT_p}{2}\right) \tag{15.50}$$

$$W_L = c_LT_L \tag{15.51}$$

and the corresponding equilibrium energy densities are:

$$W_{n0} = nw_0 = n\frac{3k_BT_L}{2} \tag{15.52}$$

$$W_{p0} = pw_0 = p\frac{3k_BT_L}{2} \tag{15.53}$$

If the hydrodynamic model is only used for one carrier, the temperature of the other carrier is set to the lattice temperature. In this case, the hydrodynamic equations are reformulated to reflect this. For example, if the hydrodynamic option is only applied to the electrons, the Joule heat generated by the hole current is directly captured in the lattice temperature equation.

4.2.4.3 Physical model parameters

The default set of transport coefficients ((Eq. 15.36) to (Eq. 15.38)) can be changed in the parameter file. The coefficients r, f^{td}, and f^{hf} are specified in the `EnergyFlux`, `ThermalDiffusion`, and `HeatFlux` sections, respectively. Energy relaxation times can be modified in the `EnergyRelaxationTime` section of the parameter file.

4.2.5 Conductivity of metals

The simulation of conductivity in metals or semi-metals is important for interconnection problems in ICs. The current density in metals is given by:

$$\vec{j}_M = -q\sigma\nabla\psi_M \tag{15.54}$$

where σ is the metal conductivity and ψ_M is the Fermi potential in the metal. For the steady state case, $\nabla\vec{j}_M = 0$. Therefore, the equation for the Fermi potential inside of metals is:

$$\nabla(\sigma\nabla\psi_M) = 0 \tag{15.55}$$

The following boundary conditions are used:

- At contacts connected to metal regions, the Dirichlet condition $\psi_M = V_{applied}$ is applied for the Fermi potential.
- Interface conditions always include the displacement current $\vec{j}_D$ in insulators and semiconductors to ensure current conservation.
- For interfaces between metal and insulator, the equations are:

$$\begin{aligned} \vec{j}_M \cdot \hat{n} &= \vec{j}_D \cdot \hat{n} \\ \psi &= \psi_M - \Phi_{MS} \end{aligned} \tag{15.56}$$

where ψ is the electrostatic potential (solution of the Poisson equation in insulator and semiconductor regions), $\Phi_{MS} = \Phi_M - \Phi_S$ is the work function difference between the metal Φ_M and an intrinsic semiconductor Φ_S selected (internally by DESSIS) as a reference material, and $\hat{n}$ is the unit normal vector to the interface. It is clear that the electrostatic potential inside metals is computed as $\psi = \psi_M - \Phi_{MS}$. This is important if, for example, there is a MOSFET with a metal gate.

For metal–semiconductor interfaces, by default, there is an Ohmic boundary condition that can be written as:

$$\begin{aligned}\vec{j}_M \cdot \vec{n} &= (\vec{j}_n + \vec{j}_p + \vec{j}_D) \cdot \vec{n} \\ \psi &= \psi_M + \psi_0 \\ n &= n_0 \\ p &= p_0\end{aligned} \quad (15.57)$$

where ψ_0 is the equilibrium electrostatic potential (the built-in potential), n_0 and p_0 are the electron and hole equilibrium concentrations (see Section 4.5.1.1 on page 15.138), and $\vec{j}_n$ and $\vec{j}_p$ are the electron and hole currents at the semiconductor side of the interface.

There are several options for the Schottky interface (see equations in Section 4.5.1.3 on page 15.140) where the potential barrier between metal and semiconductor will be computed automatically and the barrier tunneling (see Section 16.4 on page 15.306) and barrier lowering (see Section 4.5.1.4 on page 15.141) models can be applied. To select these models, use the following syntax in the input file:

```
Physics(MaterialInterface = "Metal/Silicon") { Schottky eRecVel=1e6 hRecVel=1e6 }
Physics(MaterialInterface = "Metal/Silicon") { Schottky BarrierLowering }
Physics(MaterialInterface = "Metal/Silicon") { Schottky Recombination(BarrierTunneling) }
```

For all other metal boundaries, the Neumann condition $\vec{j}_M \cdot \vec{n} = 0$ is applied.

NOTE By default, DESSIS computes output currents through a contact (on semiconductors) using doping regions, but for metals such regions do not exist (see Section 2.10.1 on page 15.73). Therefore, if the contact is located on the metal, the keyword `DirectCurrentComputationAtContact` should be specified in `Math` section. In an opposite case, the current through this contact will be zero.

Knowing that $\sigma = 1/\rho$, where ρ is the metal resistivity, the following temperature dependence is applied for ρ:

$$\rho = \rho_0(1 + \alpha_T(T - 273)) \quad (15.58)$$

All these resistivity parameters can be specified in the DESSIS parameter file as:

```
MetalResistivity {
    *   Resist(T) = Resist0 * ( 1 + TempCoef * ( T - 273 ) )
        Resist0 = <value>    # [Ohm*cm]
        TempCoef = <value>   # [1/K]
}
```

To specify the metal workfunction Φ_M, use the `Bandgap` section in the parameter file, for example:

```
Material = "Gold" {
    Bandgap{ WorkFunction = ΦM # [eV]
}
```

No specific keyword is required in the DESSIS input file because DESSIS recognizes all conductor regions and applies the appropriate equations to these regions and interfaces. The metal conductivity equation (Eq. 15.55) is a part of the `Contact` equation, which is added automatically by default. If the `NoAutomaticCircuitContact` keyword is specified in the `Math` section or only the Poisson equation is solved, the `Contact` equation should be added explicitly in the `Coupled` statement to account for conductivity in metals.

To switch off the metal conductivity, specify the following keyword in the `Math` section:

```
Math{ -MetalConductivity }
```

The thermal conductivity of metals is simulated according to the thermodynamic model with the Joule heat:

$$c\frac{\partial T}{\partial t} - \nabla \cdot \kappa \nabla T = -\nabla \psi_M \cdot \vec{j}_M \tag{15.59}$$

where κ is the lumped electron–hole–lattice thermal conductivity (see Section 24.5 on page 15.367) and c is the lattice heat capacity (see Section 24.1 on page 15.365). If it is switched off, the metal conductivity provides zero RHS of (Eq. 15.59). However, a heat flow in metals is simulated.

4.3 Quasi-Fermi potential

Electron and hole densities can be recomputed from the electron and hole quasi-Fermi potentials, and vice versa, using well-known formulas. If Boltzmann statistics are assumed, these formulas read:

$$n = N_C \exp\left(\frac{E_{F_n} - E_C}{kT}\right) \tag{15.60}$$

$$p = N_V \exp\left(\frac{E_V - E_{F_p}}{kT}\right) \tag{15.61}$$

where N_C and N_V are the effective density of states, $E_{F_n} = -q\Phi_n$ and $E_{F_p} = -q\Phi_p$ are the quasi-Fermi energies for electrons and holes, and Φ_n and Φ_p are electron and hole quasi-Fermi potentials, respectively. E_C and E_V are conduction and valence band edges, defined as:

$$E_C = -\chi + \Delta E_{g,C} - q(\psi - \psi_{ref}) \tag{15.62}$$

$$E_V = -\chi - E_g + \Delta E_{g,V} - q(\psi - \psi_{ref}) \tag{15.63}$$

where χ denotes the electron affinity and E_g the band gap.

Possible band-gap narrowing is described by $\Delta E_{g,C/V}$. Electrostatic potential ψ is computed from an arbitrarily defined reference potential ψ_{ref}. For pure materials and, in particular, for silicon, the standard approach is to set the reference potential equal to the Fermi potential of an intrinsic semiconductor. Then, (Eq. 15.60) and (Eq. 15.61) can be written as:

$$n = n_{i,eff} \cdot \exp\left(\frac{-q(\Phi_n - \psi)}{kT}\right) \tag{15.64}$$

$$p = n_{i,eff} \cdot \exp\left(\frac{q(\Phi_p - \psi)}{kT}\right) \tag{15.65}$$

where $n_{i,eff}$ is the effective intrinsic density.

In unipolar devices, such as MOSFETs, it is sometimes possible to assume that the value of quasi-Fermi potential for the minority carrier is constant in certain regions. In this case, the concentration of the minority carrier can be directly computed from (Eq. 15.64) or (Eq. 15.65). This strategy is applied in DESSIS if one of the carriers (electron or hole) is not specified inside the `Coupled` statement of the `Solve` section. DESSIS uses an approximation scheme to determine the constant value of the quasi-Fermi potential.

NOTE In many cases if avalanche generation is important, the one carrier approximation cannot be applied even for unipolar devices.

4.4 Fermi–Dirac statistics

For the equations presented in the previous sections, Boltzmann statistics for the electrons and holes were assumed. In a more general consideration, the Fermi–Dirac distribution function can be used. Fermi–Dirac statistics become important for high values of carrier densities, for example, $n > 1\times10^{19}\,/\mathrm{cm}^3$ in the active regions of a silicon device.

The Boltzmann kinetic equation with the function of the Fermi–Dirac equilibrium distribution shows that (Eq. 15.60) and for (Eq. 15.61) for electron and hole concentrations must be replaced by:

$$n = N_C F_{1/2}\left(\frac{E_{F_n} - E_C}{kT}\right) \tag{15.66}$$

$$p = N_V F_{1/2}\left(\frac{E_V - E_{F_p}}{kT}\right) \tag{15.67}$$

where $F_{1/2}$ is the Fermi integral of order 1/2.

In place of (Eq. 15.64) and (Eq. 15.65), the following equations are valid with Fermi–Dirac statistics:

$$n = n_{i,\,eff} \cdot \gamma_n \exp\left(\frac{-q(\Phi_n - \psi)}{kT}\right) \tag{15.68}$$

$$p = n_{i,\,eff} \cdot \gamma_p \exp\left(\frac{q(\Phi_p - \psi)}{kT}\right) \tag{15.69}$$

where γ_n and γ_p are the functions of η_n and η_p:

$$\gamma_n = \frac{n}{N_C}\exp(-\eta_n) \tag{15.70}$$

$$\gamma_p = \frac{p}{N_V}\exp(-\eta_p) \tag{15.71}$$

$$\eta_n = \frac{E_{F_n} - E_C}{kT_n} \tag{15.72}$$

$$\eta_p = \frac{E_V - E_{F_p}}{kT_p} \tag{15.73}$$

In the case of Fermi–Dirac statistics, important changes must be made to the current density and energy flux equations. (Eq. 15.26) and (Eq. 15.31) for the electron current density and electron energy flux density are replaced, respectively, by:

$$\vec{J}_n = \mu_n\left(qn\nabla E_C + K_B T_n \nabla n - nK_B T_n \nabla(\ln\gamma_n) + \lambda_n f_n^{td} K_B n \nabla T_n - 1.5nK_B T_n \nabla \ln m_e\right) \tag{15.74}$$

$$\vec{S}_n = -\frac{5}{2} r_n \lambda_n \left(\frac{k_B T_n}{q}\vec{J}_n + f_n^{hf}\hat{\kappa}_n \nabla T_n\right) \tag{15.75}$$

where:

$$\lambda_n = \frac{F_{1/2}(\eta_n)}{F_{-1/2}(\eta_n)} \tag{15.76}$$

Similar changes are made in (Eq. 15.27) and (Eq. 15.32) for the hole current density and hole energy flux density, respectively.

4.4.1 Syntax and implementation

To activate Fermi statistics, the keyword `Fermi` must be specified in the global `Physics` section of the input file:

```
Physics {
   ...
   Fermi
}
```

NOTE Fermi statistics can be activated only for the whole device. Region-specific or material-specific activation is not possible, and the keyword `Fermi` is ignored in any `Physics` section other than the general one. For heterostructures, the keyword `Fermi` must appear in the general `Physics` section.

4.5 Boundary conditions

4.5.1 Electrical boundary conditions

4.5.1.1 Ohmic contacts

Charge neutrality and equilibrium are assumed at the electrodes for Ohmic contacts:

$$n_0 - p_0 = N_D - N_A \tag{15.77}$$

$$n_0 p_0 = n_{i,\mathrm{eff}}^2 \tag{15.78}$$

In the case of Boltzmann statistics, this system of equations is easily solved:

$$\psi = \phi_F + \left(\frac{kT}{q}\right)\operatorname{asinh}\left(\frac{N_D - N_A}{2n_{i,\mathrm{eff}}}\right) \tag{15.79}$$

$$n_0 = \sqrt{\frac{(N_D - N_A)^2}{4} + n_{i,\mathrm{eff}}^2} + \frac{N_D - N_A}{2} \quad , \quad p_0 = \sqrt{\frac{(N_D - N_A)^2}{4} + n_{i,\mathrm{eff}}^2} - \frac{N_D - N_A}{2} \tag{15.80}$$

where n_0, p_0 are the electron and hole equilibrium concentrations, and ϕ_F is the Fermi potential at the contact that is equal to an applied voltage $V_{applied}$ if it is not a resistive contact (see Section 4.5.1.5 on page 15.141). If users select Fermi–Dirac statistics, DESSIS uses a Newton iteration scheme to obtain the equilibrium solution.

By default, $n = n_0, p = p_0$ are applied for concentrations at the Ohmic contacts. If the electron or hole recombination velocity is specified, DESSIS converts the conditions stated above to the following current boundary conditions:

$$\vec{J}_n \cdot \vec{N} = qv_n(n - n_0) \qquad \vec{J}_p \cdot \vec{N} = -qv_p(p - p_0) \tag{15.81}$$

where v_n, v_p are the electron and hole recombination velocities. In the input file, the recombination velocities can be specified as:

```
Electrode { ...
   { Name="Emitter" Voltage=0 eRecVelocity = v_n  hRecVelocity = v_p }
}
```

4.5.1.2 Gate contacts

For gate contacts, the electrostatic potential is taken as:

$$\psi = \phi_F - \Phi_{\mathrm{MS}} \tag{15.82}$$

where ϕ_F is the Fermi potential at the contact that is equal to an applied voltage $V_{applied}$ if it is not a resistive contact (see Section 4.5.1.5 on page 15.141), and $\Phi_{\mathrm{MS}} = \Phi_{\mathrm{M}} - \Phi_{\mathrm{S}}$ is the work function difference between the metal and semiconductor relative to an intrinsic semiconductor, given by the user in the `Barrier` or `WorkFunction` parameter (see Table 15.3 on page 15.39). In the `Electrode` section, the `Barrier` or `WorkFunction` can be specified as:

```
Electrode { ...
   { Name="Gate" Voltage=0 Barrier = Φ_MS }
}
Electrode { ...
   { Name="Gate" Voltage=0 WorkFunction = Φ_M }
}
```

It is possible to use a material name to define the work function, for example:

```
Electrode { ...
   { Name="Gate" Voltage=0 Material="Gold" }
}
```

In the DESSIS parameter file, the value Φ_M for the work function can be specified as:

```
Material = "Gold" {
   Bandgap { WorkFunction = ΦM }
}
```

NOTE If `Material` or `WorkFunction` is specified for an Ohmic contact that is in contact with both the semiconductor and insulator, the electrostatic potential will be equal to the built-in potential at semiconductor nodes, but for insulator nodes, it will correspond to (Eq. 15.82).

If the user needs to emulate a poly-semiconductor gate, a semiconductor material and doping concentration can be specified in the `Electrode` section:

```
Electrode { ...
   { Name="Gate" Voltage=0 Material="Silicon" (N = 5e19) }
}
```

where `N` is used to specify the doping in an n-type semiconductor material. The built-in potential is approximated by the standard expression `(kT/q)ln(N/ni)`. A p-type doping can be selected, similarly, by the letter `P`. If the value of doping concentration is not specified (only `N` or `P`), an edge of conduction or valence band is used to compute the built-in potential.

4.5.1.3 Schottky contacts

The physics of Schottky contacts is considered in detail in [4] and [197]. In this section, a typical model for Schottky contacts [92] is considered. The following boundary conditions hold:

$$\psi = \phi_F - \Phi_B + \frac{kT}{q}\ln\left(\frac{N_c}{n_{i,\mathrm{eff}}}\right) \tag{15.83}$$

$$\vec{J}_n \cdot \vec{N} = qv_n(n - n_0^B) \quad \vec{J}_p \cdot \vec{N} = -qv_p(p - p_0^B) \tag{15.84}$$

$$n_0^B = N_c\exp\left(\frac{-q\Phi_B}{kT}\right) \qquad p_0^B = N_v\exp\left(\frac{-Eg + q\Phi_B}{kT}\right) \tag{15.85}$$

where ϕ_F is the Fermi potential at the contact that is equal to an applied voltage $V_{applied}$ if it is not a resistive contact (see Section 4.5.1.5 on page 15.141), Φ_B is the barrier height (the difference between the metal work function and the electron affinity of the semiconductor), v_n and v_p are the thermionic emission velocities, and n_0^B and p_0^B are the equilibrium densities. The default values for the thermionic emission velocities v_n and v_p are 2.573 x 10^6 cm/s and 1.93 x 10^6 cm/s, respectively. The recombination velocities can be set in the `Electrode` section, for example:

```
Electrode { ...
   { Name="Gate" Voltage=0 Schottky Barrier = ΦB eRecVelocity = vn  hRecVelocity = vp  }
}
```

NOTE The `Barrier` specification can produce wrong results if the Schottky contact is connected to several different semiconductors. In such cases, use the specification `WorkFunction` = Φ_M instead of `Barrier=` Φ_B. See Section 4.5.1.2 on page 15.139 for information about gate contacts for the specification of metal materials.

4.5.1.4 Barrier lowering at Schottky contacts

The barrier lowering model can be applied to a Schottky contact. This model is useful with different physical mechanisms. The most important one is the image force [92], but the model can also be applied to some tunneling and dipole effects. The following expression is used to compute the value of the barrier lowering:

$$\Delta\Phi_B(E) = a_1\left[\left(\frac{E}{E_0}\right)^{p_1} - \left(\frac{E_{eq}}{E_0}\right)^{p_1}\right] + a_2\left[\left(\frac{E}{E_0}\right)^{p_2} - \left(\frac{E_{eq}}{E_0}\right)^{p_2}\right] \tag{15.86}$$

where E is the absolute value of the electric field [V/cm], E_0 is equal to 1 V/cm, E_{eq} is the equilibrium electric field used to have $\Delta\Phi_B = 0$ at the equilibrium, and a_1, a_2, p_1, and p_2 are model coefficients that can be specified in the DESSIS parameter file.

The final value of the Schottky barrier is computed as $\Phi_B - \Delta\Phi_B(E)$ for n-doped contacts and $\Phi_B + \Delta\Phi_B(E)$ for p-doped contacts, because a difference between the metal work function and the valence band must be considered if holes are major carriers. The barrier lowering also affects the equilibrium concentration of electrons n_0^B and holes p_0^B corresponding to its formula (Eq. 15.85).

To activate the barrier lowering model, create a new `Physics` section for each selected Schottky contact, for example:

```
Physics(Electrode = "Gate") { BarrierLowering }
```

To specify parameters of the model, create the following section in the DESSIS parameter file:

```
Electrode = "Gate"{
   BarrierLowering {
   a1 = a₁ [eV]
   p1 = p₁ [1]
   a2 = a₂ [eV]
   p2 = p₂ [1]
   }
}
```

4.5.1.5 Resistive contacts

If $V_{applied}$ is applied to the contact through a resistor R (`Resist` in the `Electrode` section) or distributed resistance R_d (`DistResist` in the `Electrode` section, see units in Section 2.3.1 on page 15.39), there is an additional equation to compute ϕ_F in (Eq. 15.79), (Eq. 15.82), and (Eq. 15.84).

For a distributed resistance, ϕ_F is different for each mesh vertex of the contact and is computed as a solution of the following equation, which is solved for each contact vertex self-consistently with the system of all equations:

$$\vec{N} \cdot (\vec{J}_p(\phi_F) + \vec{J}_n(\phi_F)) = \frac{(V_{applied} - \phi_F)}{R_d} \tag{15.87}$$

For a resistor, ϕ_F is a constant over the contact and only one of the following equations per contact is solved self-consistently with the system of all equations:

$$\int_s \vec{N} \cdot (\vec{J}_p(\phi_F) + \vec{J}_n(\phi_F)) ds = \frac{(V_{applied} - \phi_F)}{R} \tag{15.88}$$

where s is a contact area used in a DESSIS simulation to compute a total current through the contact.

NOTE In 2D simulations, s is a line along the contact. Therefore, R should be in units of Ω *<length> (Ω *µm in the input file). However, in 3D, it is Ω (see Section 2.3.1 on page 15.39).

(Eq. 15.87) and (Eq. 15.88) are written for the steady state. In the case of a transient simulation, the displacement current is added to the equations.

From emulating a behavior of a Schottky contact [173], Schottky contact resistivity at zero bias was derived and a doping-dependent resistivity model of such contacts was obtained. This model is applied to Ohmic contacts and is activated by the keyword `DistResist = SchottkyResist` in the `Electrode` section. The model is expressed as:

$$\begin{aligned} R_d &= R_\infty \exp\left(\frac{q\Phi_B}{E_0}\right) \\ E_0 &= E_{00} \cosh\left(\frac{E_{00}}{kT}\right) \\ E_{00} &= \frac{qh}{4\pi}\sqrt{\frac{|N|}{\varepsilon_s m_t}} \end{aligned} \tag{15.89}$$

where:

- Φ_B is the Schottky barrier (in this model, for electrons, this is the difference between the metal work function and the electron affinity of the semiconductor; for holes, this is the difference between the valence band energy of the semiconductor and the metal work function).
- R_∞ is the Schottky resistance for an infinite doping concentration at the contact (or zero Schottky barrier).
- N is the doping concentration equal to $N_D - N_A$ at the contact.
- ε_s is the semiconductor permittivity.
- m_t is the tunneling mass.
- T is the device lattice temperature defined in the `Physics` section (see Table 15.7 on page 15.44).

The parameters of the model can be specified in the `Electrode` section of the DESSIS parameter file. The allowed syntax and recommended default values of these parameters [173] are:

```
SchottkyResistance {
   Rinf = 2.4000e-09 , 5.2000e-09 # [Ohm*cm^2]
   PhiB = 0.6 , 0.51              # [eV]
   mt = 0.19 , 0.16               # [1]
}
```

The model is applied to each contact vertex and checks the sign of the doping concentration N. If the sign is positive, the electron parameters are used to compute R_d for the vertex. If the sign is negative, the hole parameters are used.

4.5.1.6 Floating metal gates

The charge on a floating contact (for example, a floating gate in an EEPROM cell) is specified in the `Electrode` section:

```
Electrode { ...
   { name="FloatGate" charge=1e-15 }
}
```

In the case of a floating metal gate (that is defined only by a contact with no associated semiconductor region), the electrostatic potential is determined by solving the Poisson equation with the following charge boundary condition:

$$\int \varepsilon \vec{n} \vec{\nabla} \Psi dS = Q \tag{15.90}$$

where $\vec{n}$ is the normal vector on the floating contact surface *S*, and *Q* is the specified charge on the floating contact. The electrostatic potential $\Psi_{FC} = \Psi_S$ on the floating contact surface is assumed to be constant.

An additional floating or control gate capacitance is necessary if, for example, EEPROM cells are simulated in a 2D approximation, to account for the additional influence on the capacitance from the real 3D layout. The additional floating gate capacitance can be specified in the `Electrode` statement using the keyword `FGcap`.

For example, if we have `ContGate` and `FloatGate` electrodes, additional `ContGate/FloatGate` capacitance is specified for the floating electrode:

```
Electrode {
   { name="ContGate" voltage=10 }
   { name="FloatGate" charge=0 FGcap=(value=3e-15 name="ContGate") }
}
```

where `value` is the capacitance value between `FloatGate` and the electrode `ContGate`. For the 1D case, the capacitance units are [F/μm^2]; for the 2D case, [F/μm]; and for the 3D case, [F]. The `FloatGate` electrodes can have several capacitance values for different electrodes, for example:

```
Electrode {
   { name="source" voltage=0 }
   { name="ContGate" voltage=10 }
   { name="FloatGate" charge=0 FGcap( (value=3e-15 name="ContGate") (value=2e-15 name="source") }
}
```

In the case of a transient simulation, DESSIS takes the charge specified in the `Electrode` section as an initial condition. After each time step, the charge is updated due to tunneling and hot carrier injection currents. For a description of tunneling and hot carrier injection, see Chapter 16 on page 15.299 and Chapter 17 on page 15.317.

4.5.1.7 Floating semiconductor gates

DESSIS can simulate floating semiconductor gates. Within a floating semiconductor (as opposed to metal) gate, DESSIS solves the Poisson equation with the following charge boundary condition:

$$q \int_{\mathrm{FG}} (p - n + N_D - N_A) dV = Q_{\mathrm{FG}} \tag{15.91}$$

where Q_{FG} denotes the total charge on the floating gate. The integral is calculated over all nodes of the floating gate region [127]. The charge Q_{FG} is a boundary condition that must be specified in a DESSIS `Electrode` statement in exactly the same way as for a floating metal gate:

```
Electrode {
   { name="floating_gate" Charge=1e-14 }
}
```

DESSIS automatically identifies a floating semiconductor gate based on information in the geometry (`.grd`) file. It is not necessary for the charge contact to cover the entire boundary of the floating semiconductor gate. A small contact is sufficient if the gate material has the same doping type throughout. However, if both n-type and p-type volumes exist in the gate region, separate contacts are required for each.

It is assumed that no current flows within the floating gate. Therefore, the quasi-Fermi potential for electrons and holes must be identical and constant within the floating gate:

$$\phi_n = \phi_p = \phi \tag{15.92}$$

Therefore, the electron and hole densities n and p are functions of the electrostatic potential ψ and the quasi-Fermi potential ϕ as discussed in Section 4.3 on page 15.136 and Section 4.4 on page 15.137. DESSIS does not solve the electron and hole continuity equations within a floating gate. For a steady state simulation, the charge of the floating gate must be specified as a boundary condition in the `Electrode` section. However, it is also possible to use the charge as a goal in a `Quasistationary` command:

```
Quasistationary (Goal {Name="floating_gate" Charge =1e-13})
   { Coupled {Poisson Electron} }
```

In the case of a transient simulation, DESSIS takes the charge specified in the `Electrode` section as an initial condition. After each time step, the charge is updated due to tunneling currents. For a description of tunneling models and how to enable them, see Chapter 16 on page 15.299 and Chapter 17 on page 15.317.

The floating gate capacitance can be specified in the `Electrode` statement in exactly the same way as for the floating metal gate (see Chapter 4.5.1.6 on page 15.143).

For plotting purposes, floating semiconductor gates are handled differently than other electrodes:

- The charge of the floating gate is displayed instead of the voltage.
- In the plot file, the value of the quasi-Fermi potential ϕ is used instead of the voltage.

4.5.1.8 Boundaries without contacts

All other boundaries are treated with reflective (or ideal Neumann) boundary conditions:

$$\vec{F} \cdot \vec{N} = 0 \qquad (15.93)$$

$$\vec{J}_n \cdot \vec{N} = 0 \qquad \vec{J}_p \cdot \vec{N} = 0 \qquad (15.94)$$

4.5.2 Thermal boundary conditions for thermodynamic model

For the solution of (Eq. 15.22), proper thermal boundary conditions must be applied. Wachutka [30] is followed, but the difference in thermo-powers between the semiconductor and metal at Ohmic contacts is neglected. For free, thermally insulating surfaces:

$$\kappa \frac{\partial T}{\partial N} = 0 \qquad (15.95)$$

where N denotes a unit vector in the direction of the outer normal.

At thermally conducting interfaces, thermally resistive (nonhomogeneous Neumann) boundary conditions are imposed:

$$\kappa \frac{\partial T}{\partial N} = h(T_{\mathrm{ext}} - T) \qquad (15.96)$$

where h = 1/R_{th} is the thermal surface conductance (heat transfer coefficient), which characterizes the thermal contact between the semiconductor and adjacent material, and R_{th} is the external thermal resistance. For the special case of an ideal heat sink ($h \to \infty$), Dirichlet boundary conditions are imposed:

$$T = T_{\mathrm{ext}} \qquad (15.97)$$

Dirichlet and nonhomogeneous Neumann boundary conditions (R_{th}) are specified in the DESSIS input file in the `Thermode` section (see Section 2.4 on page 15.42).

4.5.3 Thermal boundary conditions for hydrodynamic model

Boundary conditions for the lattice temperature T_L in the hydrodynamic model are specified in the same way as thermal boundary conditions for the thermodynamic model (see Section 4.5.2). For the carrier temperatures T_n and T_p, at the electrical contacts, fast relaxation to the lattice temperature (boundary condition $T_n = T_p = T_L$) is assumed.

For other boundaries, adiabatic conditions for carrier temperatures are assumed:

$$\kappa_n \frac{\partial T_n}{\partial N} = \kappa_p \frac{\partial T_p}{\partial N} = 0 \qquad (15.98)$$

4.5.4 Total thermal resistance

The thermal behavior of a semiconductor device is described in industry-standard nomenclature by the thermal resistance R_{ja} from the 'junction' or electrically active, heat-producing area of the device, to the 'ambient' or environment. The maximum device temperature at given operating conditions depends on R_{ja}. For self-heating simulations, R_{ja} can be broken down into an *internal* component that is taken into account through the thermal conductivity κ in (Eq. 15.22), and an *external* component that is defined by R_{th} and provided by the user.

The determination of R_{th} can be crucial for accurate thermal simulations since the bulk of the thermal resistance usually lies external to the usual electrical simulation domain. One approach is to simulate as much as possible of the thermal environment (die, heat sink, packaging) and estimate a reasonable external thermal resistance R_{th} based on results of thermal measurements [31]–[34] and thermal simulations [35]–[38] of semiconductor devices and packages. Including as much as possible of the thermal environment in the simulation domain has two important advantages:

- As a larger fraction of the total thermal resistance R_{ja} is accurately accounted for in the simulation domain, a smaller fraction is left for estimation through the choice of R_{th}.
- Nonuniform temperature distributions in the die, heat sink, and packaging are allowed. This permits the simulation of a realistic temperature distribution in the electrically active area for more accurate determination of self-heating effects.

The addition of large portions of the heat sink and packaging is not a problem with regard to the total number of mesh points, despite the juxtaposition of fine mesh requirements in the electrically active area with large, coarse mesh areas in the die, heat sink, and packaging. For a typical 2D mesh of approximately 1000 mesh points, such an extension of the thermal simulation domain usually adds no more than 10% to the total number of mesh points.

4.5.4.1 Estimating R_{th}

To estimate a value for R_{th}, the following must be considered:

- Bulk resistance

  The bulk thermal resistance can be calculated per unit area given the thermal conductivity κ and the length l of the material through which heat flows by the relation font $R_{th,bulk} = l/\kappa$.

- Interface resistances

  Apart from the effects of bulk thermal resistance, interfaces between materials act as barriers to heat flow. Interface thermal resistances can result from lattice mismatch, surface roughness [39], or air voids (for example, in solder joints) [40] at crystal–crystal, crystal–metal, and metal–metal interfaces. The effect of interface thermal resistances on R_{th} can be great [41].

- Radiation/convection

  The effects of heat radiation and convection can be taken into account using the linear cooling law represented by (Eq. 15.96), [42]. The value of R_{th} is appropriately reduced when radiation or convection has an important role in cooling [43]–[45].

- Thermal spreading effects

  Since the actual heat flow path in the physical device is in 3D, the value of the external resistance must be estimated taking into account the 3D geometry [46] by using an approach similar to one documented [47].

The values of R_{th} used in self-heating simulations are usually in the range $0.1 \le R_{th} \le 1.0 \ \mathrm{K W^{-1} cm^2}$.

4.5.5 Periodic boundary conditions

Periodic boundary conditions (PBCs) can be activated in DESSIS for all supported equations (see Section 4.2 on page 15.127). For ‘left’ and ‘right’ boundaries, it provides the following conditions for a selected equation:

$$\begin{aligned} \nu_{left} &= \nu_{right} \\ \overrightarrow{\Phi_{left}} &= \overrightarrow{\Phi_{right}} \end{aligned} \tag{15.99}$$

where ν is a solution variable of the selected equation (for example, the electrostatic potential ψ of the Poisson equation) and $\vec{\Phi}$ is a flux that is defined in $div\vec{\Phi}$ of the equation (see Table 15.54).

Table 15.54 Solution variables and fluxes used in periodic boundary conditions

ν	$\vec{\Phi}$	**Description**
ψ	$\varepsilon\nabla\psi$	Section 4.2.1 on page 15.128
n	$\vec{J}_n$	Section 4.2.2 on page 15.128
p	$\vec{J}_p$	
T_n	$\vec{S}_n$	Section 4.2.4 on page 15.130
T_p	$\vec{S}_p$	
T	$\kappa\nabla T$	Section 4.2.3 on page 15.128

The PBCs can be activated by the keyword `PeriodicBC(<options>)` in the `Math` section of the DESSIS input file where `<options>` provides a selection of the equations and boundaries. Multiple specifications of `<options>` are possible to select several equations: `PeriodicBC((<options1>)...(<optionsN>))`. Table 15.55 gives a complete list of possible options.

Table 15.55 Options for periodic boundary conditions

Option	**Description**
`<equation>`	Six DESSIS equation keywords (see Section 2.9 on page 15.54) are possible: `Poisson`, `Electron`, `Hole`, `eTemperature`, `hTemperature`, and `Temperature`. If no keyword is specified, the PBCs will be applied for all equations.
`Direction = <Ndir>`	`<Ndir>` is equal to 0 for the x-axis, 1 for the y-axis, and 2 for the z-axis.
`Coordinates = (<left> <right>)`	`<left>` and `<right>` are coordinates [µm] where the PBCs will be applied. If one coordinate specifies a range larger than the device size, DESSIS will take the device edges.

Table 15.55 Options for periodic boundary conditions

Option	Description
`Factor = <value>`	The default `<value>` is equal to 10^8. Usually, `Factor` should not be specified in PBCs (see the following note).

The same mesh on both sides of the device is preferable (the boundaries where the PBCs are applied), but is not necessary. It gives an additional flexibility and, for example, users can specify PBCs for one device side and for points inside the device (one of the specified `Coordinates` is inside the device).

NOTE If one side of the device differs from another (for example, oxide thickness is different), DESSIS will apply PBCs only to a part of the side that has corresponding points on another side.

An example of specifying the periodic boundary conditions for different equations and boundaries is:

```
Math{
   PeriodicBC(
      (Direction=0 Coordinates=(-1.0 2.0))
      (Poisson Direction=1 Coordinates=(-1e50 1e50))
      (Electron Direction=1 Coordinates=(-1e50 1e50))
   )
}
```

Here, the first option applies PBCs to all equations in the direction of the x-axis and for the coordinates –1.0 µm and 2.0 µm. The next two options specify PBCs for electron and hole continuity equations in the y-direction and for the device side coordinates.

NOTE If the mesh nodes are the same on PBC boundaries (the node coordinates are the same except for the axis component that corresponds to `Direction`), DESSIS will merge vertices on the 'left' and 'right' boundaries (to have one variable) and it will naturally apply PBCs where it is obvious that (Eq. 15.99) is true. If the mesh is different, (Eq. 15.99) will be provided by adding the same flux equal to $Factor(\nu_{left} - \nu_{right})$ for each mesh box (see Section 32.2 on page 15.520) on both boundaries. It will provide the flux conservation and the same ν_{left} and ν_{right} (accuracy depends on the `Factor` and a linear interpolation for ν is used if the mesh is different).

4.6 Starting solution or 'initial guess'

The starting solution or 'initial guess' is determined in DESSIS using a predetermined algorithm. An initial guess of each variable (at each mesh point) in a device required by the solution method consists of the following values:

- Electrostatic potential (Ψ)
- Quasi-Fermi potentials for electrons and holes (ϕ_n and ϕ_p) for the electrical case
- Lattice temperature (T_L) for the thermodynamic case
- Electron and hole temperatures (T_n and T_p) for the hydrodynamic case

4.6.1 Electrostatic potential and quasi-Fermi potentials: Wells

To determine an initial guess for the electrostatic potential and quasi-Fermi potentials, the device is subdivided into wells of n- and p-type doping, such that pn-junctions serve as dividers between wells. Every well is uniquely associated with a contact. The quasi-Fermi potentials in that well are set to the corresponding contact voltage, and the potentials are set to the contact voltage adjusted by the built-in voltage at the contact.

For wells that have no contacts, the following equations define the quasi-Fermi potential for the majority carriers:

$$\Phi_p = k_{Float} V_{max} + (1 - k_{Float}) V_{min} \tag{15.100}$$

$$\Phi_n = (1 - k_{Float}) V_{max} + k_{Float} V_{min} \tag{15.101}$$

where k_{Float} by default is equal to 0. The value of k_{Float} can be changed by using the keyword `FloatCoef` in the `Physics` section. For wells with more than one contact, the well is further subdivided, such that no well is associated with more than one contact.

4.6.2 Thermodynamic and hydrodynamic simulations

By default, the lattice temperature is set to 300 K throughout the device, or to the value of `Temperature` set in the `Physics` section. If the device has one or more defined thermodes, an average temperature is calculated from the temperatures at the thermodes and is set throughout the device.

As an initial guess, electron and hole temperatures are set to the lattice temperature. The electron and hole temperatures are set by initially making an estimation to the lattice temperature.

4.6.3 Save file overrides the initial guess

If created from a previous solution, the save file (with extension `.sav`) can be loaded using the keyword `Load` in the `File` section. In this case, the values of the variables in this file overwrite the initial guess values.

CHAPTER 5 Semiconductor band structure

5.1 Overview

The band gap and band edge density of states (or, in a different parameterization, the carrier effective masses, see Section 5.3 on page 15.155) are crucial parameters of a semiconductor material.

They are summarized in the intrinsic density $n_i(T)$ (for undoped semiconductors):

$$n_i(T) = \sqrt{N_C(T)N_V(T)}e^{-\frac{E_g(T)}{2k_BT}} \quad (15.102)$$

and the effective intrinsic density (including doping-dependent band-gap narrowing):

$$n_{i,\,eff} = n_i \exp\left(\frac{\Delta E_g}{2k_BT}\right) \quad (15.103)$$

In devices that contain different materials, the electron affinity χ (see Section 5.2) is also important. Along with the band gap, it determines the alignment of conduction and valence bands at material interfaces.

5.2 Band gap and electron affinity

5.2.1 Selecting a model

DESSIS supports four band gap models: `BennettWilson`, `delAlamo`, `OldSlotboom` and `Slotboom` (the same model with different parameter sets), and `TableBGN`. The band gap model can be selected in the `EffectiveIntrinsicDensity` statement in the `Physics` section of the command file, for example:

```
Physics {
   EffectiveIntrinsicDensity(BandGapNarrowing (Slotboom))
}
```

activates the `Slotboom` model. The default model is `BennettWilson`.

By default, band-gap narrowing is active. Band-gap narrowing can be switched off with the keyword `NoBandGapNarrowing`:

```
Physics {
   EffectiveIntrinsicDensity(NoBandGapNarrowing)
}
```

NOTE To plot the band gap including the band-gap narrowing, specify `EffectiveBandGap` in the `Plot` section of the command file. The quantity that DESSIS plots when `BandGap` is specified does not include band-gap narrowing.

Table 15.56 and Table 15.57 on page 15.155 list the model parameters available for calibration (see Section 5.2.2).

5.2.2 Band gap and electron affinity models

DESSIS models the lattice temperature–dependence of the band gap as [50]:

$$E_g(T) = E_g(0) - \frac{\alpha T^2}{T + \beta} \tag{15.104}$$

where T is the lattice temperature, $E_g(0)$ is the band gap energy at $0\ \text{K}$, and α and β are material parameters (see Table 15.56).

To allow $E_g(0)$ to differ for different band gap models, it is written as:

$$E_g(0) = E_{g,0} + \delta E_{g,0} \tag{15.105}$$

$E_{g,0}$ is an adjustable parameter common to all models. For the `TableBGN` model, $\delta E_{g,0} = 0$. Each of the other models offers its own adjustable parameter for $\delta E_{g,0}$ (see Table 15.56).

The effective band gap results from the band gap reduced by band-gap narrowing:

$$E_{g,\text{eff}}(T) = E_g(T) - \Delta E_g \tag{15.106}$$

The electron affinity χ is the energy separation between the conduction band and the vacuum. It is also temperature dependent and affected by band-gap narrowing:

$$\chi(T) = \chi_0 + \frac{\alpha T^2}{2(T + \beta)} + \texttt{Bgn2Chi} \cdot \Delta E_g \tag{15.107}$$

where χ_0 and `Bgn2Chi` are adjustable parameters (see Table 15.56). `Bgn2Chi` defaults to 0.5 and, therefore, band-gap narrowing splits equally between conduction and valence bands.

The main difference of the band gap models is how they handle band-gap narrowing. Band-gap narrowing in DESSIS has the form:

$$\Delta E_g = \Delta E_g^0 + \Delta E_g^{\text{Fermi}} \tag{15.108}$$

where ΔE_g^0 is determined by the particular band-gap narrowing model used, and $\Delta E_g^{\text{Fermi}}$ is an optional correction to account for carrier statistics (see (Eq. 15.112)).

5.2.2.1 Band-gap narrowing for Bennett–Wilson model

Band-gap narrowing for the Bennett–Wilson [60] model (keyword `BennettWilson`) in DESSIS reads:

$$\Delta E_g^0 = \begin{cases} E_{\text{bgn}}\left[\ln\left(\frac{N_i}{N_{\text{ref}}}\right)\right]^2 & N_i \geq N_{\text{ref}} \\ 0 & \text{otherwise} \end{cases} \tag{15.109}$$

where $N_i = N_A + N_D$ is the total doping concentration. The model was developed from absorption and luminescence data of heavily doped n-type materials. The material parameters E_{bgn} and E_{ref} are accessible in the `Bennett` parameter set in the DESSIS parameter file (see Table 15.57 on page 15.155).

5.2.2.2 Band-gap narrowing for Slotboom model

Band-gap narrowing for the Slotboom model (keyword `Slotboom` or `OldSlotboom`) (the only difference is in the parameters) in DESSIS reads:

$$\Delta E_g^0 = E_{bgn}\left[\ln\left(\frac{N_i}{N_{ref}}\right) + \sqrt{\left(\ln\left(\frac{N_i}{N_{ref}}\right)\right)^2 + 0.5}\right] \tag{15.110}$$

where $N_i = N_A + N_D$ is the total doping concentration. The models are based on measurements of $\mu_n n_i^2$ in n-p-n transistors (or $\mu_p n_i^2$ in p-n-p transistors) with different base doping concentrations and a 1D model for the collector current [51]–[54]. The material parameters E_{bgn} and E_{ref} are accessible in the `Slotboom` and `OldSlotboom` parameter sets in the DESSIS parameter file (see Table 15.57).

5.2.2.3 Band-gap narrowing for del Alamo model

Band-gap narrowing for the del Alamo model (keyword `delAlamo`) in DESSIS reads:

$$\Delta E_g^0 = \begin{cases} E_{bgn}\ln\left(\dfrac{N_i}{N_{ref}}\right) & N_i \geq N_{ref} \\ 0 & \text{otherwise} \end{cases} \tag{15.111}$$

where $N_i = N_A + N_D$ is the total doping concentration. This model was proposed [55]–[59] for n-type materials. The material parameters E_{bgn} and E_{ref} are accessible in the `delAlamo` parameter set in the DESSIS parameter file (see Table 15.57).

5.2.2.4 Table specification of band-gap narrowing

It is possible to specify band-gap narrowing by using a table, which can be defined in the parameter file in the `TableBGN` parameter set. This table gives the value of band-gap narrowing as a function of donor or acceptor concentration, or total concentration (the sum of acceptor and donor concentrations).

When specifying acceptor and donor concentrations, the total band-gap narrowing is the sum of the contributions of the two dopant types. If only acceptor or only donor entries are present in the table, the band-gap narrowing contribution for the missing dopant type vanishes. Total concentration and donor or acceptor concentration must not be specified in the same table.

Each table entry is a line that specifies a concentration type (`Donor`, `Acceptor`, or `Total`) with a concentration in cm^{-3}, and the band-gap narrowing for this concentration in eV. The actual band-gap narrowing contribution for each concentration type is interpolated from the table, using a scheme that is piecewise linear in the logarithm of the concentration.

For concentrations below (or above) the range covered by table entries, the band-gap narrowing of the entry for the smallest (or greatest) concentration is assumed, for example:

```
TableBGN {
   Total 1e16, 0
   Total 1e20, 0.02
}
```

means that for total doping concentrations below 10^{16} cm^{-3}, the band-gap narrowing vanishes, then increases up to 20 meV at 10^{20} cm^{-3} concentration and maintains this value for even greater concentrations. The interpolation is such that, in this example, the band-gap narrowing at 10^{18} cm^{-3} is 10 meV.

Tabulated default parameters for band-gap narrowing are available only for GaAs. For all other materials, users can specify the tabulated data in the parameter file.

NOTE It is not possible to specify mole fraction–dependent band-gap narrowing tables. In particular, DESSIS does not relate the parameters for ternary compound semiconductor materials to those of the related binary materials.

5.2.2.5 Band-gap narrowing with Fermi statistics

Parameters for band-gap narrowing are often extracted from experimental data assuming Maxwell–Boltzmann statistics. However, in the high-doping regime for which band-gap narrowing is important, Maxwell–Boltzmann statistics differ significantly from the more realistic Fermi statistics.

The band-gap narrowing parameters are, therefore, systematically affected by using the wrong statistics to interpret the experiment. For use in simulations that do not use Fermi statistics, this ‘error’ in the parameters is desirable, as it partially compensates the error by using the ‘wrong’ statistics in the simulation. However, for simulations using Fermi statistics, this compensation does not occur.

Therefore, DESSIS can apply a correction to the band-gap narrowing to reduce the errors introduced by using Maxwell–Boltzmann statistics for the interpretation of experiments on band-gap narrowing (see (Eq. 15.108)):

$$\Delta E_g^{\mathrm{Fermi}} = k_B 300\mathrm{K}\left[\log\left(\frac{N_V N_C}{N_A N_D}\right) + F_{1/2}^{-1}\left(\frac{N_A}{N_V}\right) + F_{1/2}^{-1}\left(\frac{N_D}{N_C}\right)\right] \tag{15.112}$$

where N_A and N_D are the acceptor and donor concentrations, N_V and N_C are the valence band and conduction band densities of states at 300 K, and $F_{1/2}^{-1}$ is the inverse of the Fermi function of order 1/2.

By default, correction (Eq. 15.112) is switched on for simulations using Fermi statistics and switched off (that is, $\Delta E_g^{\mathrm{Fermi}} = 0$) for simulations using Maxwell–Boltzmann statistics. To switch off the correction in simulations using Fermi statistics, specify `EffectiveIntrinsicDensity(NoFermi)` in the `Physics` section of the command file. This is recommended if you use parameters for band-gap narrowing that have been extracted assuming Fermi statistics. Sometimes for III–IV materials, the correction (Eq. 15.112) is too large, and it is recommended to switch it off in these cases.

5.2.3 Model parameters

The band gap $E_{g,0}$ and values of $\delta E_{g,0}$ for each model are accessible in the `BandGap` section of the parameter file, in addition to the electron affinity χ_0 and the temperature coefficients α and β. As an extension to what (Eq. 15.104) and (Eq. 15.107) suggest, the parameters $E_{g,0}$ and χ_0 can be specified at any reference temperature T_{par}. By default, they are defined at absolute zero.

Table 15.56 Band gap models: Default parameters for silicon

Symbol	Parameter name	Default	Unit	Band gap at 0 K $E_{g,0} + \delta E_{g,0} + \frac{\alpha T_{par}^2}{\beta + T_{par}}$	Reference
$E_{g,0}$	`Eg0`	1.1696	eV	–	(Eq. 15.105)
$\delta E_{g,0}$	`dEg0(Bennett)`	0.0	eV	1.1696	(Eq. 15.105)
	`dEg0(Slotboom)`	-4.795×10^{-3}	eV	1.1648	
	`dEg0(OldSlotboom)`	-1.595×10^{-2}	eV	1.1537	
	`dEg0(delAlamo)`	-1.407×10^{-2}	eV	1.1556	
α	`alpha`	4.73×10^{-4}	eV/K	–	(Eq. 15.104), (Eq. 15.107)
β	`beta`	636	K	–	(Eq. 15.104), (Eq. 15.107)
χ_0	`Chi0`	4.05	eV	–	(Eq. 15.107)
Bgn2Chi	`Bgn2Chi`	0.5	1	–	(Eq. 15.107)
T_{par}	`Tpar`	0	K	–	

Table 15.57 summarizes the silicon default parameters for the analytic band-gap narrowing models available in DESSIS.

Table 15.57 Band-gap narrowing models: Default parameters for silicon

Symbol	Parameter	Bennett	Slotboom	Old Slotboom	del Alamo	Unit
E_{bgn}	`Ebgn`	6.84×10^{-3}	6.92×10^{-3}	9.0×10^{-3}	18.7×10^{-3}	eV
N_{ref}	`Nref`	3.162×10^{18}	1.3×10^{17}	1.0×10^{17}	7.0×10^{17}	cm^{-3}

5.3 Effective masses and effective density of states

DESSIS provides two options for computing carrier effective masses and densities of states. The first method, selected by specifying `Formula=1` in the parameter file, computes an effective density of states (DOS) as a function of carrier effective mass. The effective mass may be either independent of temperature or a function of the temperature-dependent band gap. The latter is the most appropriate model for carriers in silicon and is the default for simulations of silicon devices.

In the second method, selected by specifying `Formula=2` in the parameter file the effective carrier mass is computed as a function of a temperature-dependent density of states. The default for simulations of GaAs devices is `Formula=2`.

5.3.1 Electron effective mass and DOS

5.3.1.1 Formula 1

The lattice temperature–dependence of the DOS effective mass of electrons is modeled by:

$$m_e = 6^{2/3}(m_t^2 m_l)^{1/3} + m_m \tag{15.113}$$

where the temperature-dependent effective mass component $m_t(T)$ is best described in silicon by the temperature dependence of the energy gap [48]:

$$\frac{m_t(T)}{m_0} = a \times \frac{E_g(0)}{E_g(T)} \tag{15.114}$$

The coefficient a and the mass m_1 are defined in the parameter file with the default values provided in Table 15.58 on page 15.157. The parameter m_m, which defaults to zero, allows m_e to be defined as a temperature-independent quantity if required. The variation of electron effective mass versus lattice temperature in silicon is illustrated in Figure 15.27.

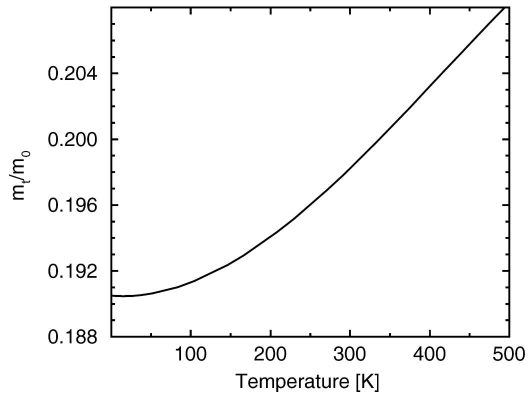


Figure 15.27 Electron transverse effective mass versus temperature

The effective densities of states (DOS) in the conduction band N_C follows from:

$$N_C(m_e, T_e) = 2.540933 \times 10^{19}\left(\frac{m_e}{m_0}\right)^{\frac{3}{2}}\left(\frac{T_e}{300}\right)^{\frac{3}{2}}\text{cm}^{-3} \tag{15.115}$$

5.3.1.2 Formula 2

If `Formula=2` is specified in the parameter file, the value for the DOS is computed from Nc (300 K), which is read from the parameter file:

$$N_C(T_e) = Nc300 \times \left(\frac{T_e}{300}\right)^{\frac{3}{2}}\text{cm}^{-3} \tag{15.116}$$

and the electron effective mass is simply a function of Nc (300 K):

$$\frac{m_e}{m_0} = \left(\frac{Nc300}{2.540 \times 10^{19}}\right)^{\frac{2}{3}} \tag{15.117}$$

5.3.2 Electron effective mass and conduction band DOS parameters

Table 15.58 lists the default coefficients for the electron effective mass and conduction band DOS models. The values can be modified in the parameter file in the `eDOSMass` section.

NOTE The default setting for the `Formula` parameter depends on the materials, for example, it is equal to 1 for silicon and 2 for GaAs.

Table 15.58 Default coefficients for effective electron mass and DOS models

Option	Symbol	Parameter name	Electrons	Unit
`Formula=1`	a	`a`	0.1905	1
	m_l	`ml`	0.9163	1
	m_m	`mm`	0	1
`Formula=2`	Nc300	`Nc300`	2.890×10^{19}	cm^{-3}

5.3.3 Hole effective mass and DOS

5.3.3.1 Formula 1

For the DOS effective mass of holes, the best fit in silicon is provided by the expression [49]:

$$\frac{m_h(T)}{m_0} = \left(\frac{a + bT + cT^2 + dT^3 + eT^4}{1 + fT + gT^2 + hT^3 + iT^4}\right)^{\frac{2}{3}} + m_m \tag{15.118}$$

where the coefficients are listed in Table 15.59 on page 15.159. The parameter m_m, which defaults to zero, allows m_h to be defined as a temperature-independent quantity.

The effective DOS for holes N_V follows from:

$$N_V(m_h, T_h) = 2.540933 \times 10^{19}\left(\frac{m_h}{m_0}\right)^{\frac{3}{2}}\left(\frac{T_h}{300}\right)^{\frac{3}{2}} \text{cm}^{-3} \tag{15.119}$$

The variation of the hole effective mass with lattice temperature in silicon is illustrated in Figure 15.28.

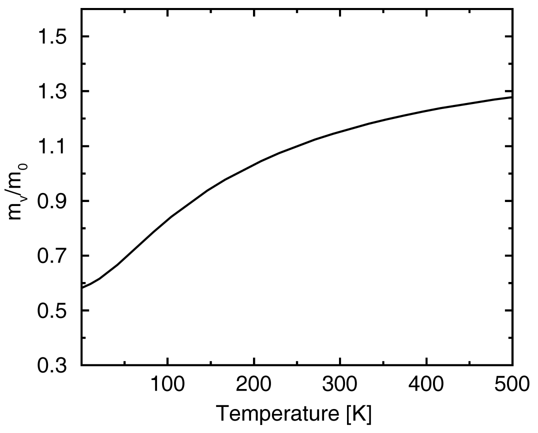


Figure 15.28 DOS hole effective mass versus temperature

5.3.3.2 Formula 2

If `Formula=2` in the parameter file, the temperature-dependent DOS is computed from N_V (300 K) as given in the parameter file:

$$N_V(T_e) = Nv300 \times \left(\frac{T_h}{300}\right)^{\frac{3}{2}} \text{cm}^{-3} \tag{15.120}$$

and the effective hole mass is given by:

$$\frac{m_h}{m_0} = \left(\frac{Nv300}{2.540 \times 10^{19}}\right)^{\frac{2}{3}} \tag{15.121}$$

5.3.4 Hole effective mass and valence band DOS parameters

The model coefficients for the hole effective mass and valence band DOS can be modified in the parameter file in the `hDOSMass` section. Table 15.59 on page 15.159 lists the default parameter values.

NOTE The default setting for the `Formula` parameter depends on the materials, for example, it is equal to 1 for silicon and it is equal to 2 for GaAs.

Table 15.59 Default coefficients for hole effective mass and DOS models

Option	Symbol	Parameter name	Electrons	Unit
`Formula=1`	a	`a`	4435870	1
	b	`b`	0.3609528×10^{-2}	K^{-1}
	c	`c`	0.1173515×10^{-3}	K^{-2}
	d	`d`	0.1263218×10^{-5}	K^{-3}
	e	`e`	0.3025581×10^{-8}	K^{-4}
	f	`f`	0.4683382×10^{-2}	K^{-1}
	g	`g`	0.2286895×10^{-3}	K^{-2}
	h	`h`	0.7469271×10^{-6}	K^{-3}
	i	`i`	0.1727481×10^{-8}	K^{-4}
	m_m	`mm`	0	1
`Formula=2`	Nv300	`Nv300`	$3.140\text{x}10^{19}$	cm^{-3}

CHAPTER 6 Incomplete ionization

6.1 Overview

In silicon, with the exception of indium, dopants can be considered to be fully ionized at room temperature because the impurity levels are sufficiently shallow. However, when impurity levels are relatively deep compared to the thermal energy $(k_B T)/q$ at room temperature, incomplete ionization must be considered. This is the case for indium acceptors in silicon and nitrogen donors and aluminum acceptors in silicon carbide. In addition, for simulations at reduced temperatures, incomplete ionization must be considered for all dopants. For these situations, DESSIS has an ionization probability model based on activation energy. The ionization (activation) is computed separately for each species present.

DESSIS supports the most significant dopants used in silicon technologies: the donors As, P, Sb, and N, and the acceptors B and In. For the simulation of other semiconductors such as III–V compounds, the actual donors and acceptors used (Si, Be, and so on) are not supported. However, it is reasonable to replace the real dopant species with an appropriate silicon donor or acceptor and adjust the physical parameters accordingly.

For example, to simulate GaAs, which is doped n-type using Si, P can be substituted as the donor. The donor level and cross-sections of P should then be changed in the parameter file to represent the values for silicon in GaAs. Alternatively, DESSIS supports the generic dopants 'NDopant' and 'PDopant.'

The doping profiles for these generic dopants are read from the data file under the names `NDopantConcentration` and `PDopantConcentration`.

6.2 Syntax and implementation

The incomplete ionization model is activated with the keyword `IncompleteIonization` in the `Physics` section of the input file:

```
Physics{ IncompleteIonization }
```

The incomplete ionization model for selected species is activated with the additional keyword `Dopants`:

```
Physics{ IncompleteIonization(Dopants = "Species_name1 Species_name2 ...") }
```

For example, the statement `Physics{ IncompleteIonization(Dopants = "BoronActiveConcentration") }` activates the incomplete ionization model only for boron.

The incomplete ionization model can be specified in region or material physics (see Section 2.5.3 on page 15.47). In this case, the model is activated only in these regions or materials.

6.3 Physical model description

The concentration of ionized impurity atoms is given by Fermi–Dirac distribution:

$$N_D^+ = \frac{N_D}{1 + g_D \exp\left(\frac{E_{F_n} - E_D}{k_B T}\right)} \quad \text{for } N_D < N_{D,crit} \tag{15.122}$$

$$N_A^- = \frac{N_A}{1 + g_A \exp\left(\frac{E_A - E_{F_p}}{k_B T}\right)} \quad \text{for } N_A < N_{A,crit} \tag{15.123}$$

where $N_{D/A}$ is the substitutional (active) dopant concentration, $g_{D/A}$ is the degeneracy factor for the impurity level, and $E_{D/A}$ is the donor/acceptor ionization (activation) energy.

In the literature [179], incomplete ionization in SiC material has been considered and another general distribution function has been proposed, which can be expressed as:

$$N_D^+ = \frac{N_D}{1 + G_D(T) \exp\left(\frac{E_{F_n} - E_C}{k_B T}\right)} \tag{15.124}$$

$$N_A^- = \frac{N_A}{1 + G_A(T) \exp\left(-\frac{E_{F_p} - E_V}{k_B T}\right)} \tag{15.125}$$

where $G_D(T)$ and $G_A(T)$ are the ionization factors discussed in [180][181]. These factors can defined by a PMI (see [179] and Section 33.27.3 on page 15.596). By comparing (Eq. 15.122) and (Eq. 15.123) with (Eq. 15.124) and (Eq. 15.125), it can be seen that, in the case of the Fermi–Dirac distribution function, the ionization factors can be written as:

$$G_D(T) = g_D \cdot \exp\left(\frac{\Delta E_D}{k_B T}\right), \ \Delta E_D = E_C - E_D \ \text{ and } \ G_A(T) = g_A \cdot \exp\left(\frac{\Delta E_A}{k_B T}\right), \ \Delta E_A = E_A - E_V \tag{15.126}$$

In DESSIS, the basic variables are potential, electron concentration, and hole concentration. Therefore, it is more convenient to rewrite (Eq. 15.122) and (Eq. 15.123) in terms of the carrier concentration instead of the quasi-Fermi levels:

$$N_D^+ = \frac{N_D}{1 + g_D \frac{n}{n_1}}, \text{ with } \quad n_1 = N_C \cdot \exp\left(-\frac{\Delta E_D}{k_B T}\right) \quad \text{for } N_D < N_{D,crit} \tag{15.127}$$

$$N_A^- = \frac{N_A}{1 + g_A \frac{p}{p_1}}, \text{ with } \quad p_1 = N_V \cdot \exp\left(-\frac{\Delta E_A}{k_B T}\right) \quad \text{for } N_A < N_{A,crit} \tag{15.128}$$

The expressions for n_1 and p_1 in these two equations are valid for Boltzmann statistics and without quantization. If Fermi–Dirac statistics or a quantization model (see Chapter 7 on page 15.165) is used, n_1 and p_1 are multiplied by the coefficients γ_n and γ_p defined in (Eq. 15.70) and (Eq. 15.71) (see Section 4.4 on page 15.137). For $N_{D/A} > N_{D/A,crit}$, the dopants are assumed to be completely ionized, in which case, every donor and acceptor species is considered in the Poisson equation. The values of $N_{D,crit}$ and $N_{A,crit}$ can be adjusted in the DESSIS parameter file.

The donor and acceptor activation energies are effectively reduced by the total doping in the semiconductor. This effect is accounted for in the expressions:

$$\Delta E_D = \Delta E_{D,0} - \alpha_D \cdot N_i^{1/3} \tag{15.129}$$

$$\Delta E_A = \Delta E_{A,0} - \alpha_A \cdot N_i^{1/3} \tag{15.130}$$

where $N_i = N_A + N_D$ is the total doping concentration. In transient simulations, the terms:

$$\frac{\partial N_D^+}{\partial t} = \sigma_D v_{\mathrm{th}}^n \left[\frac{n_1}{g_D} N_D - \left(n + \frac{n_1}{g_D}\right) N_D^+\right] \tag{15.131}$$

$$\frac{\partial N_A^-}{\partial t} = \sigma_A v_{\mathrm{th}}^p \left[\frac{p_1}{g_A} N_A - \left(p + \frac{p_1}{g_A}\right) N_A^-\right] \tag{15.132}$$

are included in the continuity equations ($v_{\mathrm{th}}^{n,p}$ denote the carrier thermal velocities).

6.4 Physical model parameters

The values of the dopant level $E_{A/D,0}$, the doping-dependent shift parameter $\alpha_{A/D}$, the impurity degeneracy factor $g_{A/D}$, and the cross section $\sigma_{A/D}$ are accessible in the parameter file in the `Ionization` section.

Table 15.60 Default coefficients for incomplete ionization model for dopants in silicon

Symbol	Parameter name	Default value for species ☼								Unit
		As	P	Sb	B	In	N	NDopant	PDopant	
$E_{A/D,0}$	E_☼_0	0.054	0.045	0.039	0.045	0.16	0.045	0.045	0.045	eV
$\alpha_{A/D}$	alpha_☼	3.1×10^{-8}								eV.cm
$g_{A/D}$	g_☼	2	2	2	4	4	2	2	4	1
$\sigma_{A/D}$	Xsec_☼	1.0×10^{-12}								cm^2/s
$N_{D,crit}$	NdCrit	1.0×10^{22}								cm^{-3}
$N_{A,crit}$	NaCrit	1.0×10^{22}								cm^{-3}

For each user-defined dopant (see Section 2.14.2 on page 15.98), the `Ionization` section must contain a separate subsection where the parameters $E_{A/D,0}$, $\alpha_{A/D}$, $g_{A/D}$, and $\sigma_{A/D}$ are defined. For example, for the dopant `Nitrogen` described in Section 2.14.2 for the material SiC, the subsection can be written as:

```
Material = "SiC" {
...
       Ionization {
       ...
          Species("Nitrogen") {
             type   = donor
             E_0    = 0.1
             alpha  = 0
             g      = 2
             Xsec   = 1.0e-15
          }
       }
}
```

The field `type` can be omitted because the dopant type is specified in the `datexcodes.txt` file. It is used here for informative purposes only.

6.5 Example: Incomplete ionization

The ionization of nitrogen donors and aluminum acceptors in SiC has been modeled under equilibrium conditions at temperatures of 300 K and 800 K. The results are plotted in Figure 15.29. For n-type SiC, the nitrogen donor level had the values $E_C - E_D = 100$ meV and $g_D = 2$. The p-type SiC results were calculated assuming the aluminum acceptor values $E_A - E_V = 200$ meV and $g_A = 4$.

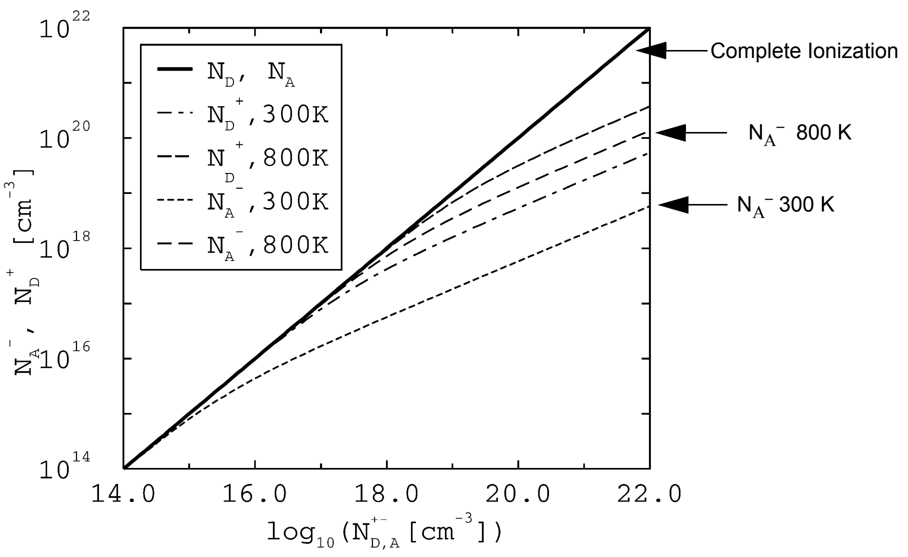


Figure 15.29 Incomplete ionization in n-type (N donor) and p-type (Al acceptor) SiC at 300 K and 800 K

CHAPTER 7 Quantization models

7.1 Overview

The scaling rules for modern submicron devices require a thinner oxide and higher level of channel doping. Some features of current MOSFETs (oxide thickness, channel width) have reached quantum mechanical length scales. Therefore, the wave nature of electrons and holes can no longer be neglected. The most basic quantization effects in MOSFETs are the shift of the threshold voltage and reduction of the gate capacity.

To include quantization effects in a classical device simulation, a simple approach is to introduce an additional potential-like quantity Λ in the classical density formula, which reads:

$$n = N_{\mathrm{C}} \exp\left(\frac{E_{\mathrm{F}_n} - E_{\mathrm{C}} - \Lambda}{k_{\mathrm{B}} T}\right) \tag{15.133}$$

where n is the electron density, T is the carrier temperature, k_{B} is the Boltzmann constant, N_{C} is the conduction band density of states, E_{C} is the conduction band energy, and E_{F_n} is the electron Fermi energy. (For brevity, only the formulas for electrons are given; holes are handled analogously.) When using Fermi statistics, the exponential function in (Eq. 15.133) is replaced by a Fermi integral of order $1/2$.

The most important effects related to the density modification (due to quantization) can be captured by proper models for Λ. Other effects (for example, single electron effects) exceed the scope of this approach.

DESSIS implements three quantization models, that is, three different models for Λ. They differ in physical sophistication, numeric expense, and robustness:

- The van Dort quantum correction model (see Section 7.2 on page 15.166) is a numerically robust, fast, and proven model. It is only suited to MOSFET simulations. Structures such as quantum wells and SOI transistors with ultrathin silicon layer (below approximately 10 nm) are beyond the scope of this model. While important terminal characteristics are well described by this model, it does not give the correct density distribution in the channel.
- The 1D Schrödinger equation (see Section 7.3 on page 15.167) is the most physically sophisticated quantization model. It can be used for MOSFET simulation, and quantum well and ultrathin SOI simulation. Simulations with this model tend to be slow and often lead to convergence problems, which restrict its use to situations with small current flow. Therefore, the Schrödinger equation is used mainly for the validation and calibration of other quantization models.
- The density gradient model (see Section 7.4 on page 15.172) is numerically robust, but significantly slower than the van Dort model. It can be applied to MOSFETs, quantum wells and SOI structures, and gives a reasonable description of terminal characteristics and charge distribution inside a device. Compared to the other quantization models, it can describe 2D and 3D quantization effects.

7.2 van Dort quantum correction model

7.2.1 Model description

To account for quantization in MOSFET channels, the van Dort quantum correction model [112] is implemented in DESSIS.

The van Dort model computes Λ of (Eq. 15.133) as a function of E_n, the electric field normal to the semiconductor–insulator interface:

$$\Lambda = \frac{13}{9} \cdot k_{\text{fit}} \cdot F(\bar{d}) \cdot \left(\frac{\varepsilon\varepsilon_0}{4kT}\right)^{1/3} \cdot \left|(E_n - E_{\text{crit}})\right|^{2/3} \tag{15.134}$$

where k_{fit} and E_{crit} are fitting parameters. The function $F(\bar{d})$ is defined by:

$$F(\bar{d}) = \frac{2 \cdot \exp(-a^2(\vec{r}))}{1 + \exp(-2a^2(\vec{r}))} \tag{15.135}$$

where $a(\vec{r}) = l(\vec{r})/\lambda_{\text{ref}}$ and $l(\vec{r})$ is a distance from the point $\vec{r}$ to the interface. The parameter λ_{ref} effectively defines the region near the interface where quantum correction occurs. Depending on the sign of the normal electric field, band gap widening is applied to the conduction band or valence band. Assuming E_n is the electric field pointing outside the semiconductor, band gap widening is applied to the holes (valence band correction) if $E_n > 0$, or the electrons (conduction band correction) if $E_n < 0$ (that is, quantum correction is applied to those carriers that are drawn towards the interface by the electric field; for the carriers driven away from the interface, the correction is 0). The van Dort model recognizes all semiconductor–insulator interfaces regardless of their orientation.

7.2.2 Syntax and implementation

To activate the model, the parameter `QCvanDort` must be specified in the `Physics` section. If the quantum mechanical band gap widening is taken into account only for electrons (holes), the parameter `eQCvanDort` (`hQCvanDort`) is used:

```
Physics { ... QCvanDort}
```

Table 15.61 lists the coefficients for this model. The default value of λ_{ref} is from the literature [112]. Other parameters were obtained by fitting the model to experimental data for electrons [112] and holes [113]. The van Dort model parameters can be adjusted and are accessible in the `vanDortQMModel` section of the parameter file.

Table 15.61 Default coefficients for van Dort quantum correction model

Symbol	Electrons		Holes		Unit
k_{fit}	`eFit`	2.4e–08	`hFit`	1.8e–08	eV · cm
E_{crit}	`eEcritQC`	1e+05	`eEcritQC`	1e+05	(V)/(cm)
λ_{ref}	`dRef`	2.5e–06	`dRef`	2.5e–06	cm

By default, E_n is electric field normal to the semiconductor–insulator interface. If required, this interface can be redefined by specifying `EnormalInterface` explicitly in the `Math` section (see Section 5.2.1 on page 15.151).

7.3 One-dimensional Schrödinger solver

The 1D Schrödinger solver implements the most physically sophisticated quantization model in DESSIS. To use the Schrödinger solver:

1. Construct a special purpose 'nonlocal' line mesh (see Section 7.3.1).
2. Activate the Schrödinger solver on the nonlocal line mesh with appropriate parameters (see Section 7.3.2 on page 15.168).

Sometimes, especially for heteromaterials, the physical model parameters need to be adapted (see Section 7.3.3 on page 15.169).

NOTE The 1D Schrödinger solver is time consuming and often causes converge problems. Furthermore, small-signal analysis (see Section 3.8.3 on page 15.117) and noise and fluctuation analysis (see Chapter 15 on page 15.291) are not possible when using this solver.

7.3.1 Defining a nonlocal line mesh

The specification of the nonlocal line mesh determines where in the device the 1D Schrödinger equation can be solved. To generate the nonlocal line mesh, specify the `NonLocal` keyword in a contact-specific or an interface-specific `Math` section of the DESSIS command file.

Options to `NonLocal` control the construction of the nonlocal line mesh. For example:

```
Math(RegionInterface="gateoxide/channel") {
        NonLocal(
            Length=10-e7
            Permeation=1e-7
            Direction=(0 1 0) MaxAngle=5
        )
}
```

generates a nonlocal line mesh at the interface between region `gateoxide` and `channel`. The nonlocal mesh lines extend 10 nm (according to `Length`) to one side of the interface and 1 nm (according to `Permeation`) to the other side. In this example, `Direction` and `MaxAngle` restrict the nonlocal line mesh to lines that run along the y-axis, with a tolerance of 5°. `Direction` defines a vector that is typically perpendicular to the interface; therefore, the specification above is appropriate for an interface in the x-z plane.

Additional options to `NonLocal` can be specified for region-specific `Math` sections. For nonlocal meshes constructed for the Schrödinger equation, the `-Transparent` option is important. For example:

```
Math(Region="gateoxide"){
    Nonlocal(-Transparent)
}
```

This specification with the interfacewise specification suppresses the construction of nonlocal lines for which the part with length `Length` goes through `gateoxide`. This means that nonlocal mesh lines extend 10 nm into the channel and 1 nm into the gateoxide region, and not the other way round. Assuming gateoxide is at least 1 nm thick, the nonlocal lines do not extend into a poly gate that may be located on top of gateoxide and, therefore, the 1D Schrödinger equation will not be solved in the poly gate.

Typically, all NonLocal options demonstrated in these two examples are required (and sufficient) to obtain a proper nonlocal line mesh for the 1D Schrödinger equation. For more information about constructing nonlocal meshes, see Section 2.10.7 on page 15.83.

7.3.2 Activating and controlling the 1D Schrödinger solver

To activate the 1D Schrödinger for electrons (and holes), specify the Electron (Hole) option to the Schroedinger keyword in the Physics section for the interface or contact for which the nonlocal line mesh was specified. For example:

```
Physics(RegionInterface="gateoxide/channel"){
   Schroedinger(Electron)
}
```

activates the Schrödinger equation for electrons on the nonlocal line mesh constructed for the interface between the gateoxide and channel regions.

Additional options to Schroedinger determine numeric accuracy and which eigenstates will be computed. Table 15.62 lists the optional keywords.

Table 15.62 Options for Schroedinger keyword

Keyword	Description
Electron	Switches on Schrödinger equation for electrons.
-Electron	Switches off Schrödinger equation for electrons. This is the default.
Hole | -Hole	Corresponding keywords for holes.
MaxSolutions	The maximum number of eigensolutions computed per non-equivalent set of band valleys. The latter term refers to materials such as silicon, which have heavy-hole and light-hole bands and six conduction band valleys. The default is 5.
Error	The precision to which the eigenenergies are computed [eV]. The default is 1e–5.
EnergyInterval	The highest energy to which eigensolutions are computed. EnergyInterval is given in eV and measured from the lowest interior potential point on the nonlocal line on which the Schrödinger equation is solved. When the value is the default (0), only bound solutions are computed.
Direction = <integer>	Only used in backward compatibility mode. Direction in which carriers are quantized (1 = x-direction, 2 = y-direction, 3 = z-direction). The default is 1.
SubstOrient = <x,y,z>	Only used in backward compatibility mode. x, y, and z are numbers that give the vector in reciprocal space that corresponds to the quantization direction. This is used for materials such as silicon that have nonisotropic band minima. The default is (1,0,0).

The options MaxSolutions, Error, and EnergyInterval have an optional argument electron or hole. For example, the specification MaxSolutions=30 sets the value for both electrons and holes. However, the specifications MaxSolutions(electron)=2 and MaxSolutions(hole)=3 set different values for electrons and holes.

For backward compatibility, the Schrödinger equation can be solved without the specification of a nonlocal line mesh. For the backward compatibility mode, activate the Schrödinger equation regionwise in parts of the device where quantization is important. The simulator automatically extracts grid lines from those regions. Each grid line spans as many regions as possible, that is, all adjacent regions for which the Schrödinger equation is activated. Typically, each grid line spans at least two regions. For each region belonging to the same grid line, all parameters set in the command file must agree.

Restrictions to the use of the Schrödinger equation in the backward compatibility mode are:

- A tensor grid is required in the part of the device where the Schrödinger equation is solved, with axes parallel to the main axes (x, y, z).
- No region interfaces are allowed in this part except when perpendicular to the quantization direction.

7.3.3 Physical parameters

Parameters used to solve the Schrödinger equation are redefined in the `SchroedingerParameters` section in the DESSIS parameter file. The `formula` parameter pair specifies which expressions for the masses are used in the Schrödinger equation. For each carrier, if the formula is `0`, DESSIS uses the isotropic density-of-states mass. For electrons, if the formula is `1`, DESSIS uses the anisotropic (silicon-like) masses specified in the `eDOSMass` parameter set. For holes, ‘wrapped’ band structure (formula `1`), or heavy-hole or light-hole masses (formula `2`) are available (see Table 15.63).

It is impossible to solve the Schrödinger equation across semiconductors with fundamentally different band structure. DESSIS displays an error message if band structure compatibility is violated.

For materials with anisotropic band extrema, the `LatticeParameters` parameter set is used to determine the crystal orientation with respect to the mesh axes (see Section 20.1.1 on page 15.339).

In addition to the mass parameters, `SchroedingerParameters` offers a parameter `offset` to model the lifting of the degeneracy of band minima in strained materials. The parameter `offset` is a pair of an electron and a hole value. If the value for electrons is positive, DESSIS adds the value to the band edge for the valley of lower degeneracy, but leaves the band edge for the valley with higher degeneracy unaffected. If the electron value is negative, DESSIS subtracts the value from the band edge for the valley of higher degeneracy (but leaves the valley with lower degeneracy unaffected).

For holes, DESSIS subtracts positive values from the band edge for the heavy-hole band and adds negative values to the band edge of the light-hole band (in both cases, DESSIS leaves the other band edge unaffected). The signs are such that the shift always moves band edges outwards, away from midgap. The gap (defined as the minimum separation of the band edges) remains unchanged.

Table 15.63 Coefficients for hole effective masses in Schrödinger solver

Formula	Symbol	Parameter name	Default value	Unit
`1`	A	`A`	4.22	1
	B	`B`	0.6084	1
	C	`C`	23.058	1
`2`	m_1	`ml`	0	1
	m_h	`mh`	0	1

7.3.4 Visualizing the results

To visualize the results obtained by the Schrödinger equation, DESSIS offers special keywords for the `NonLocalPlot` section (see Section 2.10.7.3 on page 15.85).

To plot the wavefunctions, specify the `WaveFunction` keyword in the `NonLocalPlot` section. DESSIS plots wavefunctions in units of $(\mu\text{m})^{-1/2}$. To plot the eigenenergies, specify the `EigenEnergy` keyword; DESSIS plots them in units of eV. The names of the curves in the output have two numeric indices. The first index denotes the valley index and the second index denotes the number of zeros of the wavefunction (see ν and j in (Eq. 15.136)).

Without further specification, DESSIS will plot all eigenenergies and wavefunctions it has computed. The `Electron` and `Hole` options to the `WaveFunction` and `EigenEnergy` keywords restrict the output to the wavefunctions and eigenenergies for electrons and holes, respectively. The `Electron` and `Hole` options have a sub-option `Number=<num>` to restrict the output to data for the *num* states of lowest energy. For example:

```
NonLocal(
    (0 0)
){
   WaveFunction(Electron(Number=3) Hole)
}
```

will plot all hole wavefunctions and the three lowest-energy electron wavefunctions for the nonlocal mesh line close to the coordinate $(0, 0, 0)$.

7.3.5 Model description

The 1D Schrödinger equation is:

$$\left(-\frac{\partial}{\partial z}\frac{\hbar^2}{2m_{z,\nu}(z)}\frac{\partial}{\partial z}+E_C(z)\right)\Psi_{j,\nu}(z) = E_{j,\nu}\Psi_{j,\nu}(z) \qquad (15.136)$$

where z is the quantization direction along which the Schrödinger equation is solved (typically, the direction perpendicular to the silicon–oxide interface in a MOSFET), $\hbar = h/2\pi$ is the reduced Planck constant; E_C is the (position-dependent) conduction band energy, ν labels the band valley (for example, in silicon, there are six conduction band valleys), $m_{z,\nu}$ is the (position-dependent) effective mass component for valley ν in quantization direction, $\Psi_{j,\nu}$ is the j-th normalized eigenfunction in valley ν, and $E_{j,\nu}$ is the j-th eigenenergy.

From the solution of this equation, the density is computed as:

$$n(z) = \frac{k_B T(z)}{\pi\hbar^2}\sum_{j,\nu}|\Psi_{j,\nu}(z)|^2 m_{xy,\nu}(z)\exp\left(\frac{E_{F_n}(z)-E_{j,\nu}}{kT(z)}\right) \qquad (15.137)$$

where $m_{xy,\nu}$ is the mass component perpendicular to the quantization direction for valley ν. Equating (Eq. 15.137) to (Eq. 15.133) gives an expression for Λ. For simplicity, the coordinates x and y in all function arguments are omitted. For Fermi statistics, the $\exp(\ldots)$ in (Eq. 15.137) is replaced by $\log(1+\exp(\ldots))$.

The Schrödinger equation is solved over a finite domain $[z_-, z_+]$. At the endpoints of this domain, the boundary condition:

$$\frac{\Psi'_{j,\nu}}{\Psi_{j,\nu}} = \mp\frac{\sqrt{2m_{z,\nu}|E_{j,\nu} - E_C|}}{\hbar} \tag{15.138}$$

is applied, where for the upper sign, all position-dependent functions are taken at z_+ and, for the lower sign, at z_- .

For electrons in materials with anisotropic effective mass, the quantization mass is computed as:

$$\frac{1}{m_{z,\nu}} = \sum_{i=1}^{3}\frac{z_i^2}{m_{i,\nu}} \tag{15.139}$$

where z_i are the coefficients of the unit normal vector pointing in the z -direction, expressed in a coordinate system, where the main axes (labelled by i) are associated to the main axes of the crystal structure, and $m_{i,\nu}$ are the effective mass components for these main axes. If the band structure is isotropic, the respective mass directly determines the quantization mass.

For hole masses, there is a choice between a model for ‘wrapped’ hole bands and a simpler model with two isotropic bands. The former is the default for silicon; the latter is the default for compound semiconductors such as GaAs, AlAs, and InGaAs.

In the case of ‘warped’ hole bands, the quantization mass is given by:

$$m_{z,\nu} = \frac{m_0}{A \pm \sqrt{B + C\cdot\left((z_1z_2)^2 + (z_2z_3)^2 + (z_3z_1)^2\right)}} \tag{15.140}$$

where m_0 is the free electron mass. The coefficients A , B , and C are accessible in the parameter file (see Table 15.63 on page 15.169). The positive sign is for the light-hole band and the negative sign is for the heavy-hole band. If isotropic hole masses are selected, the heavy-hole mass m_h and light-hole mass m_l specified in the parameter file directly give $m_{z,\nu}$.

The masses $m_{xy,\nu}$ are chosen to achieve the same density-of-state mass as used in classical simulations:

$$m_{xy,\nu} = \frac{{m_{\mathrm{DOS},1}}^{3/2}}{m_{z,\nu}^{1/2}} \tag{15.141}$$

where the per-band density-of-states mass $m_{\mathrm{DOS},1}$ is obtained from the total density-of-states mass m_{DOS} and the number of band valleys n_{valleys} by the relation $n_{\mathrm{valleys}}{m_{\mathrm{DOS},1}}^{3/2} = m_{\mathrm{DOS}}^{3/2}$.

Strain can lift the degeneracy of conduction and valence band valleys. Therefore, in (Eq. 15.136), in addition to the effective mass, the band edge depends on ν , that is, $E_C \rightarrow E_{C,\nu}$. To account for this effect, DESSIS has the parameter `offset` that determines these band-edge shifts for electrons and holes. For the interpretation of this parameter, see Section 7.3.3 on page 15.169.

7.3.6 Application notes

- Convergence problems: Near the flat band condition, minor changes in the band structure can change the number of bound states between zero and one. By default, DESSIS computes only bound states and, therefore, this change switches from a purely classical solution to a solution with quantization. To avoid the large change in density that this transition can cause, set the variable `EnergyInterval` to a nonzero value (for example, 1). Then, a few unbound states are computed and a hard transition is avoided.
- Always include a part of the 'barrier' in the nonlocal line on which the Schrödinger equation is solved. In a MOSFET, besides the channel region, solve the Schrödinger equation in a part of the oxide adjacent to the channel (for example, 1 nm deep). Use the `Permeation` option to `NonLocal` to achieve this (see Section 7.3.1 on page 15.167). Failure to solve for a part of the oxide adjacent to the channel blinds the Schrödinger solver to the barrier. It does not know the electrons are confined and the required quantization effects are not obtained.

7.4 Density gradient model

7.4.1 Model description

For the density gradient model [150][151], Λ in (Eq. 15.133) is given in terms of a partial differential equation:

$$\Lambda = -\frac{\gamma\hbar^2}{12m}\left\{\nabla^2\log n + \frac{1}{2}(\nabla\log n)^2\right\} = -\frac{\gamma\hbar^2}{6m}\frac{\nabla^2\sqrt{n}}{\sqrt{n}} \tag{15.142}$$

where $\hbar = h/2\pi$ is the reduced Planck constant, m is the DOS mass, and γ is a fit factor.

Introducing the reciprocal thermal energy $\beta = 1/k_B T$, the mass driving term $\Phi_m = -k_B T\log(N_C/N_{ref})$ (with the conduction band edge DOS N_C and an arbitrary normalization constant N_{ref}), the electrostatic potential ψ, the conduction band energy E_C, and the smoothed potential $\Phi = E_C + \Phi_m + \Lambda$, (Eq. 15.142) can be rewritten as:

$$\begin{aligned}\Lambda = -\frac{\hbar^2\gamma}{12m}\{\nabla\cdot(\xi\nabla\beta E_{F_n} - \nabla\beta\bar{\Phi} + (\eta-1)q\nabla\beta\psi) + \\ \vartheta(\xi\nabla\beta E_{F_n} - \nabla\beta\bar{\Phi} + (\eta-1)q\nabla\beta\psi)^2\} + \Lambda_{PMI}\end{aligned} \tag{15.143}$$

with the parameters $\xi = \eta = 1$ and $\vartheta = 1/2$. These parameters are used to investigate alternative formulations of the quantum potential corrections [152]. In insulators, DESSIS does not compute the Fermi energy; therefore, in insulators, $\xi = \eta = 0$. By default, DESSIS uses (Eq. 15.143), which for Fermi statistics deviates from (Eq. 15.142), even when $\xi = \eta = 1$ and $\vartheta = 1/2$. The density-based expression (Eq. 15.142) is available as an option (see Section 7.4.2 on page 15.173).

In (Eq. 15.143), the quantity Λ_{PMI} is computed from a user-specified PMI model (see Section 33.15 on page 15.569). By default, $\Lambda_{PMI} = 0$.

At Ohmic contacts, resistive contacts, and current contacts, the boundary condition $\Lambda = 0$ is imposed.

At Schottky contacts, gate contacts, interfaces to metal regions, and external boundaries, homogeneous Neumann boundary conditions are used, $\hat{n} \cdot (\xi \nabla \beta E_{\mathrm{F}_n} - \nabla \beta \overline{\Phi} + (\eta - 1) q \nabla \beta \psi) = 0$, with $\hat{n}$ the normal vector on the boundary. At internal interfaces, Φ and $\hat{n} \cdot (\xi \nabla \beta E_{\mathrm{F}_n} - \nabla \beta \overline{\Phi} + (\eta - 1) q \nabla \beta \psi)$ must be continuous.

Optionally, DESSIS offers a modified mobility to improve the modeling of tunneling through semiconductor barriers:

$$\mu = \frac{\mu_{\mathrm{cl}} + r\mu_{\mathrm{tunnel}}}{1 + r} \tag{15.144}$$

where μ_{cl} is the usual (classical) mobility as described in Chapter 8 on page 15.175, μ_{tunnel} is a fit parameter, and $r = max(0, n/n_{\mathrm{cl}} - 1)$. Here, n_{cl} is the 'classical' density (see (Eq. 15.220)). In this modification, for $n > n_{\mathrm{cl}}$, the additional carriers (density $n - n_{\mathrm{cl}}$) are considered as tunneling carriers that are subject to a different mobility μ_{tunnel} than the classical carriers (density n_{cl}).

7.4.2 Syntax and implementation

The density gradient quantum corrections are switched on in the `Physics` section by using the keywords `eQuantumPotential` and `hQuantumPotential` for the corrections for electrons and holes, respectively. These keywords can also be used in regionwise or materialwise `Physics` sections. The corrections can be switched off by `-eQuantumPotential` and `-hQuantumPotential`.

Even when the correction is not activated for the entire device, Λ is computed in all insulator and semiconductor regions. However, in (Eq. 15.133), Λ is taken into account only where the quantum corrections are activated. In metal regions, the quantum potential is never computed. Semiconductor–metal and insulator–metal interfaces are handled like external boundaries.

The option `Density` to `eQuantumPotential` and `hQuantumPotential` activates the density-based formula (Eq. 15.142) instead of the potential-based formula (Eq. 15.143). If this option is used, the parameters ξ and η must both be 1.

A string (that is, a name enclosed in double quotation marks) as the option to `eQuantumPotential` and `hQuantumPotential` specifies the name of a PMI model for Λ_{PMI} in (Eq. 15.143) (see Chapter 33 on page 15.535). When no PMI model is specified, $\Lambda_{\mathrm{PMI}} = 0$.

Apart from switching on the quantum corrections in the `Physics` section, the equations for the quantum corrections must be solved by using `eQuantumPotential` or `hQuantumPotential`, or both in the `Solve` section. For example:

```
Physics {
   eQuantumPotential
}
Plot {
   eQuantumPotential
}
Solve {
   Coupled { Poisson eQuantumPotential }
   Quasistationary (
      DoZero InitialStep=0.01 MaxStep=0.1 MinStep=1e-5
      Goal { Name="gate" Voltage=2 }
   ){
      Coupled { Poisson Electron eQuantumPotential }
   }
}
```

The quantum corrections can be plotted. To this end, use `eQuantumPotential` or `hQuantumPotential`, or both in the `Plot` section.

To activate the mobility modification according to (Eq. 15.144), specify the `Tunneling` option to the keyword `Mobility` in the `Physics` section of the DESSIS command file. Specify μ_{tunnel} for electrons and holes by the `mutunnel` parameter pair in the `ConstantMobility` parameter set of the DESSIS parameter file.

The parameters γ, ϑ, ξ, and η are available in the parameter file in the parameter set `QuantumPotentialParameters`. The parameters can be specified regionwise and materialwise. They cannot be functions of the mole fraction in heterodevices. In insulators, ξ is always assumed as zero, regardless of user specifications.

7.4.3 Application notes

- Fitting parameters: The parameters (γ) for the density gradient model have been calibrated only for silicon. The quantum correction affects the densities and field distribution in a device. Therefore, parameters for mobility and recombination models that have been calibrated to classical simulations (or simulations with the van Dort model) may require recalibration.
- Tunneling: The density gradient model increases the current through the semiconducting potential barriers. However, this effect is not a trustworthy description of tunneling through the barrier. To model tunneling, use one of the dedicated models that DESSIS provides (see Chapter 16 on page 15.299). To suppress unwanted tunneling or to fit tunneling currents despite these concerns, consider using the modified mobility model according to (Eq. 15.144).
- Convergence: In general and particularly for the density gradient corrections, solving additional equations worsens convergence. Typically, it is advisable to solve the equations for the quantum potentials whenever the Poisson equation is solved (using a `Coupled` statement). Usually, the best strategy to obtain an initial solution at the beginning of the simulation is to do a coupled solve of the Poisson equation and the quantum potentials, without the current and temperature equations.

  To obtain this initial solution, it is sometimes necessary that the user sets the `LineSearchDamping` optional parameter (see Section 2.9.1 on page 15.55) to a value less than 1; a good value is 0.01.

  Often, using initial bias conditions that induce a current flow work well for a classical simulation, but do not work when the density gradient model is active. In such cases, start from equilibrium bias conditions and put an additional voltage ramping at the beginning of the simulation.

  If an initial solution is still not possible, consider using Fermi statistics (see Section 4.4 on page 15.137). Check grid refinement and pay special attention to interfaces between insulators and highly doped semiconductor regions. For classical simulations, the refinement perpendicular to such interfaces is often not critical, whereas for quantum mechanical simulations, quantization introduces variations at small length scales, which must be resolved.

CHAPTER 8 Mobility models

8.1 Overview

DESSIS uses a modular approach for the description of the carrier mobilities. In the simplest case, the mobility is a function of the lattice temperature. This so-called constant mobility model described in Section 8.3 on page 15.176 should only be used for undoped materials. For doped materials, the carriers scatter with the impurities. This leads to a degradation of the mobility. Section 8.4 on page 15.176 introduces the models that describe this effect.

Models that describe the mobility degradation at interfaces, for example, the silicon–oxide interface in the channel region of a MOSFET, are introduced in Section 8.5 on page 15.180. These models account for the scattering with surface phonons and surface roughness.

Models that describe the effects of carrier–carrier scattering are given is Section 8.6 on page 15.186. The Philips unified mobility model described in Section 8.7 on page 15.188 is a well-calibrated model, which accounts for both impurity and carrier–carrier scattering. Finally, the models that describe mobility degradation in high electric fields are discussed in Section 8.8 on page 15.193.

8.2 Syntax and implementation

The mobility models are selected in the `Physics` section as arguments of the `Mobility` keyword:

```
Physics{ Mobility( <arguments> ) ...}
```

If more than one mobility model is activated, the different mobility contributions are combined according to the following scheme – different bulk (μ_{b1}, μ_{b2}, ...) and surface (μ_{s1}, μ_{s2}, ...) mobility contributions are combined following Mathiessen's rule:

$$\frac{1}{\mu} = \frac{1}{\mu_{b1}} + \frac{1}{\mu_{b2}} + \ldots + \frac{1}{\mu_{s1}} + \frac{1}{\mu_{s2}} + \ldots \qquad (15.145)$$

If the high field saturation model is activated, the final mobility is computed in two steps. First, the low field mobility μ_{low} is determined according to (Eq. 15.145). Second, the final mobility is computed from a (model-dependent) formula as a function of a driving force F:

$$\mu = f(\mu_{\text{low}}, F) \qquad (15.146)$$

8.3 Mobility due to lattice scattering

8.3.1 Syntax and implementation

The constant mobility model is switched on by default in DESSIS. It is active unless the keyword `DopingDependence` is specified as an argument of the keyword `Mobility`:

```
Physics{ Mobility( DopingDependence ) ...}
```

8.3.2 Constant mobility model

The constant mobility model [61] assumes that carrier mobility is only affected by phonon scattering and, therefore, dependent only on the lattice temperature:

$$\mu_{\mathrm{const}} = \mu_{\mathrm{L}}\left(\frac{T}{T_0}\right)^{-\zeta} \tag{15.147}$$

where μ_{L} is the mobility due to bulk phonon scattering, T is the lattice temperature, and $T_0 = 300\ \mathrm{K}$. The default values of μ_{L} and the exponent ζ are listed in Table 15.64.

8.3.3 Constant mobility model parameters

The constant mobility model parameters are accessible in the parameter file in the `ConstantMobility:{...}` section.

Table 15.64 Constant mobility model: Default coefficients for silicon

Symbol	Parameter name	Electrons	Holes	Unit
μ_{L}	`mumax`	1417	470.5	$\mathrm{cm}^2/(\mathrm{Vs})$
ζ	`exponent`	2.5	2.2	1

8.4 Doping-dependent mobility degradation

In doped semiconductors, scattering of the carriers by charged impurity ions leads to degradation of the carrier mobility. DESSIS supports two models for doping-dependent mobility.

8.4.1 Syntax and implementation

The models for the mobility degradation due to impurity scattering are activated by specifying the `DopingDependence` arguments for the keyword `Mobility`:

```
Physics{ Mobility( DopingDependence ...) ...}
```

Different models are available and are selected by an additional argument:

```
Physics{ Mobility( DopingDependence( [ Masetti | Arora | UniBo ] ) ...) ...}
```

If the keyword `DopingDependence` is specified without additional keywords, DESSIS uses a material-dependent default. For example, in silicon, the default is the `Masetti` model; for GaAs, the default is the Arora model. The default model (Masetti or Arora) for each material can be specified by the variable `formula`, which is accessible in the `DopingDependence` section of the parameter file:

```
DopingDependence: {
   formula= 1 , 1    # [1]
... }
```

If the variable `formula` is set to 1, the Masetti model is selected. To activate the Arora model, set the variable `formula` to 2. The `DopingDependence` parameter set also determines the other parameters for the Masetti and Arora models. The parameters for the University of Bologna model are in the `UniBoDopingDependence` parameter set:

```
UniBoDopingDependence:{...}
```

8.4.2 Masetti model

The default model used by DESSIS to simulate doping-dependent mobility in silicon was proposed by Masetti et al. [62]:

$$\mu_{\text{dop}} = \mu_{\min 1}\exp\left(-\frac{P_c}{N_{\text{i}}}\right) + \frac{\mu_{\text{const}} - \mu_{\min 2}}{1+\left(\dfrac{N_{\text{i}}}{C_r}\right)^{\alpha}} - \frac{\mu_1}{1+\left(\dfrac{C_s}{N_{\text{i}}}\right)^{\beta}} \tag{15.148}$$

where $N_{\text{i}} = N_{\text{A}} + N_{\text{D}}$ denotes the total concentration of ionized impurities.

The reference mobilities $\mu_{\min 1}$, $\mu_{\min 2}$, and μ_1, the reference doping concentrations P_c, C_r, and C_s, and the exponents α and β are accessible in the parameter file in the section:

```
DopingDependence:{ ... }
```

The corresponding values for silicon are given in Table 15.65.

Table 15.65 Masetti model: Default coefficients

Symbol	Parameter name	Electrons	Holes	Unit
$\mu_{\min 1}$	`mumin1`	52.2	44.9	cm^2/(Vs)
$\mu_{\min 2}$	`mumin2`	52.2	0	cm^2/(Vs)
μ_1	`mu1`	43.4	29.0	cm^2/(Vs)
P_c	`Pc`	0	9.23×10^{16}	cm^{-3}
C_r	`Cr`	9.68×10^{16}	2.23×10^{17}	cm^{-3}
C_s	`Cs`	3.34×10^{20}	6.10×10^{20}	cm^{-3}
α	`alpha`	0.680	0.719	1
β	`beta`	2.0	2.0	1

The low-doping reference mobility μ_{const} is determined by the constant mobility model (see Section 8.3 on page 15.176).

Figure 15.30 illustrates the doping-dependent mobility degradation in silicon for electrons and holes.

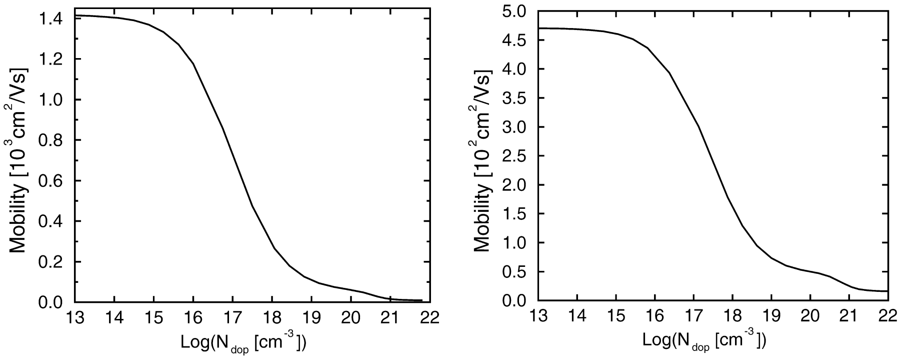


Figure 15.30   Doping dependence of mobility in silicon: Electrons (*left*) and holes (*right*) according to (Eq. 15.150)

8.4.3   Arora model

The Arora model [117] reads:

$$\mu_{dop} = \mu_{min} + \frac{\mu_d}{1 + \left(\frac{N_i}{N_0}\right)^{A*}} \tag{15.149}$$

with:

$$\mu_{min} = A_{min} \cdot \left(\frac{T}{T_0}\right)^{\alpha_m} , \quad \mu_d = A_d \cdot \left(\frac{T}{T_0}\right)^{\alpha_d} \tag{15.150}$$

and:

$$N_0 = A_N \cdot \left(\frac{T}{T_0}\right)^{\alpha_N} , \quad A* = A_a \cdot \left(\frac{T}{T_0}\right)^{\alpha_a} \tag{15.151}$$

where $N_{\text{i}} = N_{\text{A}} + N_{\text{D}}$ denotes the total concentration of ionized impurities, T_0 = 300 K, and T is the lattice temperature. All other parameters are accessible in the parameter file in the `DopingDependence` section.

Table 15.66   Arora model: Default coefficients for silicon

Symbol	Parameter name	Electrons	Holes	Unit
A_{min}	`Ar_mumin`	88	54.3	cm^2/(Vs)
α_{m}	`Ar_alm`	–0.57	–0.57	1
A_d	`Ar_mud`	1252	407	cm^2/(Vs)
α_d	`Ar_ald`	–2.33	–2.23	1
A_N	`Ar_N0`	1.25×10^{17}	2.35×10^{17}	cm^{-3}

Table 15.66 Arora model: Default coefficients for silicon

Symbol	Parameter name	Electrons	Holes	Unit
α_N	`Ar_alN`	2.4	2.4	1
A_a	`Ar_a`	0.88	0.88	1
α_a	`Ar_ala`	–0.146	–0.146	1

8.4.4 University of Bologna bulk mobility model

The University of Bologna bulk mobility model was developed for an extended temperature range between 25°C and 400°C. It should be used together with the University of Bologna inversion layer mobility model (see Section 8.5.3 on page 15.183). The model [144][145] is based on the Masetti approach with two major extensions. First, attractive and repulsive scattering are separately accounted for, therefore, leading to a function of both donor and acceptor concentrations. This automatically accounts for different mobility values for majority and minority carriers, and ensures continuity at the junctions as long as the impurity concentrations are continuous functions. Second, a suitable temperature dependence for most model parameters is introduced to predict correctly the temperature dependence of carrier mobility in a wider range of temperatures, with respect to other models. The temperature dependence of lattice mobility is reworked, with respect to the default temperature.

The model for lattice mobility is:

$$\mu_L(T) = \mu_{\max}\left(\frac{T}{300\text{ K}}\right)^{-\gamma + c\left(\frac{T}{300\text{ K}}\right)} \tag{15.152}$$

where $\mu_{\max}$ denotes the lattice mobility at room temperature, and c gives a correction to the lattice mobility at higher temperatures. The maximum mobility $\mu_{\max}$ and the exponents γ and c are accessible in the parameter file.

The model for bulk mobility reads:

$$\mu_{\text{dop}}(N_D, N_A, T) = \mu_0(N_D, N_A, T) + \frac{\mu_L(T) - \mu_0(N_D, N_A, T)}{1 + \left(\frac{N_D}{C_{r1}(T)}\right)^{\alpha} + \left(\frac{N_A}{C_{r2}(T)}\right)^{\beta}} - \frac{\mu_1(N_D, N_A, T)}{1 + \left(\frac{N_D}{C_{s1}(T)} + \frac{N_A}{C_{s2}(T)}\right)^{-2}} \tag{15.153}$$

In turn, μ_0 and μ_1 are expressed as weighted averages of the corresponding limiting values for pure acceptor-doping and pure donor-doping densities:

$$\mu_0(N_D, N_A, T) = \frac{\mu_{0d}N_D + \mu_{0a}N_A}{N_D + N_A} \tag{15.154}$$

$$\mu_1(N_D, N_A, T) = \frac{\mu_{1d}N_D + \mu_{1a}N_A}{N_D + N_A} \tag{15.155}$$

The reference mobilities μ_{0d}, μ_{0a}, μ_{1d}, and μ_{1a}, and the reference doping concentrations C_{r1}, C_{r2}, C_{s1}, and C_{s2} are accessible in the parameter file.

Table 15.67 lists the corresponding values of silicon for arsenic, phosphorus, and boron; $T_n = T/300\text{ K}$.

Table 15.67 Parameters of University of Bologna bulk mobility model

Silicon	Parameter name	Electrons (As)	Electrons (P)	Holes (B)	Unit
μ_{max}	`mumax`	1441	1441	470.5	cm^2/(Vs)
c	`Exponent2`	0.07	0.07	0	1
γ	`Exponent`	2.45	2.45	2.16	1
μ_{0d}	`mumin1`	$55.0T_n^{-\gamma_{0d}}$	$62.2T_n^{-\gamma_{0d}}$	$90.0T_n^{-\gamma_{0d}}$	cm^2/(Vs)
γ_{0d}	`mumin1_exp`	0.6	0.7	1.3	1
μ_{0a}	`mumin2`	$132.0T_n^{-\gamma_{0a}}$	$132.0T_n^{-\gamma_{0a}}$	$44.0T_n^{-\gamma_{0a}}$	cm^2/(Vs)
γ_{0a}	`mumin2_exp`	1.3	1.3	0.7	1
μ_{1d}	`mu1`	$42.4T_n^{-\gamma_{1d}}$	$48.6T_n^{-\gamma_{1d}}$	$28.2T_n^{-\gamma_{1d}}$	cm^2/(Vs)
γ_{1d}	`mu1_exp`	0.5	0.7	2.0	1
μ_{1a}	`mu2`	$73.5T_n^{-\gamma_{1a}}$	$73.5T_n^{-\gamma_{1a}}$	$28.2T_n^{-\gamma_{1a}}$	cm^2/(Vs)
γ_{1a}	`mu2_exp`	1.25	1.25	0.8	1
C_{r1}	`Cr`	$8.9 \times 10^{16} T_n^{\gamma_{r1}}$	$8.5 \times 10^{16} T_n^{\gamma_{r1}}$	$1.3 \times 10^{18} T_n^{\gamma_{r1}}$	cm^{-3}
γ_{r1}	`Cr_exp`	3.65	3.65	2.2	1
C_{r2}	`Cr2`	$1.22 \times 10^{17} T_n^{\gamma_{r2}}$	$1.22 \times 10^{17} T_n^{\gamma_{r2}}$	$2.45 \times 10^{17} T_n^{\gamma_{r2}}$	cm^{-3}
γ_{r2}	`Cr2_exp`	2.65	2.65	3.1	1
C_{s1}	`Cs`	$2.9 \times 10^{20} T_n^{\gamma_{s1}}$	$4.0 \times 10^{20} T_n^{\gamma_{s1}}$	$1.1 \times 10^{18} T_n^{\gamma_{s1}}$	cm^{-3}
γ_{s1}	`Cs_exp`	0	0	6.2	1
C_{s2}	`Cs2`	7.0×10^{20}	7.0×10^{20}	6.1×10^{20}	cm^{-3}
α	`alpha`	0.68	0.68	0.77	1
β	`beta`	0.72	0.72	0.719	1

NOTE The DESSIS default parameters for electrons are those for arsenic. The bulk mobility model was calibrated with experiments [145] in the temperature range 300–700 K. It is suitable for isothermal simulations at large temperatures or nonisothermal simulations.

8.5 Mobility degradation at interfaces

In the channel region of a MOSFET, the high transverse electric field forces carriers to interact strongly with the semiconductor–insulator interface. Carriers are subjected to scattering by acoustic surface phonons and surface roughness. The models in this section describe mobility degradation caused by these effects.

8.5.1 Syntax and implementation

To activate mobility degradation at interfaces, select a method to compute the transverse field $F_\perp$ (see Section 8.5.4 on page 15.185).

To select the calculation of field perpendicular to the semiconductor–insulator interface, specify the keyword `Enormal` in the `Mobility` statement:

```
Physics{ Mobility( Enormal ...) ...}
```

Alternatively, to select calculation of $F_\perp$ perpendicular to current flow, specify:

```
Physics{ Mobility( ToCurrentEnormal ...) ...}
```

To select a mobility degradation model, specify an option to `Enormal` or `ToCurrentEnormal`. Valid options are `Lombardi`, `UniBo`, or a PMI model provided by the user. For example,

```
Physics{ Mobility( Enormal(UniBo) ...) ...}
```

selects the University of Bologna model (see Section 8.5.3 on page 15.183). The default model is `Lombardi` (see Section 8.5.2).

NOTE The mobility degradation models discussed in this section are very sensitive to mesh spacing. ISE recommends that the vertical mesh spacing be reduced to 0.1 nm in the silicon at the oxide interface underneath the gate. For the extensions of the Lombardi model (see (Eq. 15.159)), even smaller spacing of 0.05 nm is appropriate. This fine spacing is only required in the two uppermost rows of mesh and can be increased progressively moving away from the interface.

8.5.2 Enhanced Lombardi model

The surface contribution due to acoustic phonon scattering has the form:

$$\mu_{ac} = \frac{B}{F_\perp} + \frac{C(N_i/N_0)^\lambda}{F_\perp^{1/3}(T/T_0)^k} \tag{15.156}$$

and the contribution attributed to surface roughness scattering is given by:

$$\mu_{sr} = \left(\frac{(F_\perp/F_{ref})^{A*}}{\delta} + \frac{F_\perp^3}{\eta}\right)^{-1} \tag{15.157}$$

These surface contributions to the mobility (μ_{ac} and μ_{sr}) are then combined with the bulk mobility μ_b according to Mathiessen's rule (see Section 8.3 on page 15.176 and Section 8.4 on page 15.176):

$$\frac{1}{\mu} = \frac{1}{\mu_b} + \frac{D}{\mu_{ac}} + \frac{D}{\mu_{sr}} \tag{15.158}$$

In the above formulas, $N_i = N_A + N_D$ refers to the total concentration of ionized impurities and $T_0 = 300\text{ K}$. The reference field $F_{ref} = 1\text{ V/cm}$ ensures a unitless numerator in (Eq. 15.157). $F_\perp$ is the transverse electric field normal to the semiconductor–insulator interface, see Section 8.5.4. $D = \exp(-x/l_{crit})$ (where x is the distance from the interface and l_{crit} a fit parameter) is a damping that switches off the inversion layer terms far away from the interface. All other parameters are accessible in the parameter file.

In the Lombardi model [61], the exponent A^* in (Eq. 15.157) is equal to 2. According to another study [116], an improved fit to measured data is achieved if A^* is given by:

$$A^* = A + \frac{\alpha_{\perp}(n+p)N_{\mathrm{ref}}^{\nu}}{(N_{\mathrm{i}}+N_1)^{\nu}} \qquad (15.159)$$

where n and p denote the electron and hole concentrations, respectively. The reference doping concentration $N_{\mathrm{ref}} = 1\ \mathrm{cm}^{-3}$ cancels the unit of the term raised to the power ν in the denominator of (Eq. 15.159). The Lombardi model parameters are accessible in the DESSIS parameter file in the section:

```
EnormalDependence:{ ... }
```

The respective default parameters that are appropriate for silicon are given in Table 15.68. The parameters B, C, N_0, and λ were fitted at SGS Thomson and are not contained in the literature [61].

Table 15.68 Lombardi model: Default coefficients for silicon

Symbol	Parameter name	Electrons	Holes	Unit
B	B	4.75×10^{7}	9.925×10^{6}	cm/s
C	C	5.80×10^{2}	2.947×10^{3}	$\mathrm{cm}^{5/3}/(\mathrm{V}^{2/3}\mathrm{s})$
N_0	N0	1	1	cm^{-3}
λ	lambda	0.1250	0.0317	1
k	k	1	1	1
δ	delta	5.82×10^{14}	2.0546×10^{14}	$\mathrm{cm}^2/(\mathrm{Vs})$
A	A	2	2	1
$\alpha_{\perp}$	alpha	0	0	cm^3
N_1	n1	1	1	cm^{-3}
ν	nu	1	1	1
η	eta	5.82×10^{30}	2.0546×10^{30}	$\mathrm{V}^2/(\mathrm{cms})$
l_{crit}	l_crit	1×10^{-6}	1×10^{-6}	cm

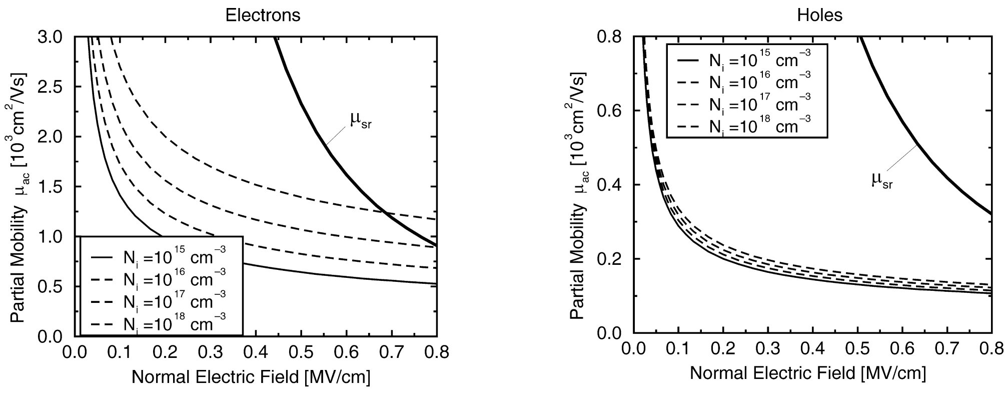


Figure 15.31 Partial mobilities μ_{ac} and μ_{sr} of Lombardi model as function of normal electric field for different doping concentrations

The modifications of the Lombardi model suggested in [116] can be activated by setting $\alpha_\perp$ to a nonzero value. Table 15.69 lists a consistent set of parameters for the modified model.

Table 15.69 Lombardi model: Lucent coefficients for silicon

Symbol	Parameter name	Electrons	Holes	Unit
B	`B`	3.61×10^7	1.51×10^7	cm/s
C	`C`	1.70×10^2	4.18×10^3	cm$^{(5/3)}$/(V$^{(2/3)}$s)
N_0	`N0`	1	1	cm^{-3}
λ	`lambda`	0.0233	0.0119	1
k	`k`	1.7	0.9	1
δ	`delta`	3.58×10^{18}	4.10×10^{15}	cm^2/(Vs)
A	`A`	2.58	2.18	1
$\alpha_\perp$	`alpha`	6.85×10^{-21}	7.82×10^{-21}	cm^3
N_1	`n1`	1	1	cm^{-3}
ν	`nu`	0.0767	0.123	1
η	`eta`	1×10^{50}	1×10^{50}	V^2/(cms)
l_{crit}	`l_crit`	1	1	cm

8.5.3 University of Bologna inversion layer mobility model

The University of Bologna inversion layer mobility model was developed for an extended temperature range between 25°C and 400°C. It should be used together with the University of Bologna bulk mobility model (see Section 8.4.4 on page 15.179). The inversion layer mobility in MOSFETs is degraded by Coulombic scattering at low normal fields and by surface phonons and surface roughness scattering at large normal fields.

In the University of Bologna model [144], all these effects are combined by using Mathiessen's rule:

$$\frac{1}{\mu} = \frac{1}{\mu_{\text{bsc}}} + \frac{D}{\mu_{\text{ac}}} + \frac{D}{\mu_{\text{sr}}} \tag{15.160}$$

where $1/\mu_{\text{bsc}}$ is the contribution of Coulombic scattering, and $1/\mu_{\text{ac}}$, $1/\mu_{\text{sr}}$ are those of surface phonons and surface roughness scattering, respectively. $D = \exp(-x/l_{\text{crit}})$ (where x is the distance from the interface and l_{crit} a fit parameter) is a damping that switches off the inversion layer terms far away from the interface.

The term μ_{bsc} is associated with substrate impurity and carrier concentration. It is decomposed in an unscreened part (due to the impurities) and a screened part (due to local excess carrier concentration):

$$\mu_{\text{bsc}} = \mu_{\text{b}}[D(1+f_{\text{sc}}^{\tau})^{1/\tau} + (1-D)] \tag{15.161}$$

where μ_{b} is given by the bulk mobility model, and τ is a fit parameter. The screening function is given by:

$$f_{\text{sc}} = \left(\frac{N_1}{N_{\text{i}}}\right)^{\eta} \frac{N_{\text{min}}}{N_{\text{i}}} \tag{15.162}$$

where $N_{\text{i}} = N_{\text{A}} + N_{\text{D}}$ is the total concentration of ionized impurities and N_{min} is the minority carrier concentration.

If surface mobility is plotted against the effective normal field, mobility data converges toward a universal curve. Deviations from this curve appear at the onset of weak inversion, and the threshold field changes with the impurity concentration at the semiconductor surface [146]. The term μ_{bsc} models these deviations, in that, it is the roll-off in the effective mobility characteristics. As the effective field increases, the mobilities become independent of the channel doping and approach the universal curve.

The main scattering mechanisms are, in this case, surface phonons and surface roughness scattering, which are expressed by:

$$\mu_{\text{ac}} = C(T)\left(\frac{N_{\text{i}}}{N_2}\right)^{a} \frac{1}{F_{\perp}^{\delta}} \tag{15.163}$$

$$\mu_{\text{sr}} = B(T)\left(\frac{N_{\text{i}}+N_3}{N_4}\right)^{b} \frac{1}{F_{\perp}^{\lambda}} \tag{15.164}$$

All model parameters are accessible in the parameter file. $F_{\perp}$ is the electric field normal to semiconductor–insulator interface, see Section 8.5.4 on page 15.185. Table 15.70 lists the respective silicon default parameters.

Table 15.70 Parameters of University of Bologna inversion layer mobility model ($T_{\text{n}} = T/300\text{ K}$)

Symbol	Parameter name	Electrons	Holes	Unit
N_1	`N1`	2.34×10^{16}	2.02×10^{16}	cm^{-3}
N_2	`N2`	4.0×10^{15}	7.8×10^{15}	cm^{-3}
N_3	`N3`	1.0×10^{17}	2.0×10^{15}	cm^{-3}
N_4	`N3`	2.4×10^{18}	6.6×10^{17}	cm^{-3}

Table 15.70 Parameters of University of Bologna inversion layer mobility model ($T_n = T/300\text{ K}$)

Symbol	Parameter name	Electrons	Holes	Unit
B	`B`	$5.8 \times 10^1\ T_n^{\gamma_B}$	$7.82 \times 10^{15}\ T_n^{\gamma_B}$	cm^2/(Vs)
γ_B	`B_exp`	0	1.4	1
C	`C`	$1.86\times10^4 T_n^{-\gamma_C}$	$5.726\times10^3 T_n^{-\gamma_C}$	cm^2/(Vs)
γ_C	`C_exp`	1.6	1.3	1
τ	`tau`	1	3	1
η	`eta`	0.3	0.5	1
a	`ac_exp`	0.026	–0.02	1
b	`sr_exp`	0.11	0.08	1
l_{crit}	`l_crit`	1×10^{-6}	1×10^{-6}	cm
δ	`delta`	0.29	0.3	1
λ	`lambda`	2.64	2.24	1

NOTE The reported parameters are detailed in the literature [147]. The model was calibrated with experiments [146][147] in the temperature range 300–700 K.

The parameters for the model are accessible in the parameter file section:

```
UniBoEnormalDependence:{...}
```

8.5.4 Transverse field computation

DESSIS supports two different methods for computing the normal electric field $F_\perp$.

8.5.4.1 Normal to the interface

Assume that mobility degradation occurs at an interface Γ. In this method, $F_\perp(\vec{r})$ is determined for a given point $\vec{r}$ in the device by locating the nearest point $\vec{r}_i$ at the interface Γ, then determining the direction $\vec{e}_\perp$ normal to the interface point $\vec{r}_i$, and then computing the component of the electric field in this direction at the point $\vec{r}$:

$$F_\perp(\vec{r}) = \left|\vec{F}(\vec{r}) \bullet \vec{e}_\perp\right| \tag{15.165}$$

To activate the Lombardi model with this method of computing $F_\perp$, specify the argument `Enormal` in the `Mobility` statement. The keyword `ToInterfaceEnormal` is synonymous with `Enormal`.

By default, the interface Γ is a semiconductor–insulator interface. Sometimes, it is important to change this default interface definition, for example, when the insulator (oxide) is considered a wide band gap semiconductor. In this case, DESSIS allows this interface to be specified explicitly in the `Math` section (see Section 5.2.1 on page 15.151).

In the following example, DESSIS takes as Γ the union of interfaces between materials, `OxideAsSemiconductor` and `Silicon`, and regions, `regionK1` and `regionL1`:

```
Math {
   ...
EnormalInterface(regioninterface=["regionK1/regionL1"],
                 materialinterface=["OxideAsSemiconductor/Silicon"])
   ...
}
```

8.5.4.2 Normal to the current flow

Using this method, $F_{\perp}(\vec{r})$ is defined as the component of the electric field normal to the electron (c=e) and hole (c=h) currents $\vec{j}_c(\vec{r})$:

$$F_{c,\perp}(\vec{r}) = |\vec{F}(\vec{r})|\sqrt{1-\left(\frac{\vec{F}(\vec{r})\bullet\vec{j}_c(\vec{r})}{|\vec{F}(\vec{r})|\cdot|\vec{j}_c(\vec{r})|}\right)^2} \tag{15.166}$$

where $F_{e,\perp}$ is used for the evaluation of electron mobility and $F_{h,\perp}$ is used for the evaluation of hole mobility.

NOTE For very low current levels, the computation of the electric field component normal to the currents may be numerically problematic and lead to convergence problems. It is recommended to use the option `Enormal`. Besides possible numeric problems, both approaches give the same or very similar results.

8.6 Carrier–carrier scattering

DESSIS supports two models for the description of carrier–carrier scattering. One model is based on Choo [64] and Fletcher [65] and uses the Conwell–Weisskopf screening theory. As an alternative to the Conwell–Weisskopf model, DESSIS supports the Brooks–Herring model. The carrier–carrier contribution to the overall mobility degradation is captured in the mobility term μ_{eh}. This is combined with the mobility contributions from other degradation models (μ_{other}) according to Mathiessen's rule:

$$\frac{1}{\mu} = \frac{1}{\mu_{other}} + \frac{1}{\mu_{eh}} \tag{15.167}$$

8.6.1 Syntax and implementation

The carrier–carrier scattering models are activated by specifying the `CarrierCarrierScattering` argument for the `Mobility` keyword. Either of the two different models are selected by an additional argument:

```
Physics{ Mobility(
   CarrierCarrierScattering( [ ConwellWeisskopf | BrooksHerring ] )
   ...)...}
```

The Conwell–Weisskopf model is the default in DESSIS for carrier–carrier scattering and is activated when the keyword `CarrierCarrierScattering` is specified without any arguments.

8.6.2 Conwell–Weisskopf model

$$\mu_{eh} = \frac{D\left(\frac{T}{T_0}\right)^{3/2}}{\sqrt{np}}\left[\ln\left(1+F\left(\frac{T}{T_0}\right)^2(pn)^{-1/3}\right)\right]^{-1} \tag{15.168}$$

where n and p are the electron and hole densities respectively, T denotes the lattice temperature, and T_0 =300 K. The parameters D and F are accessible in the parameter file. The default values appropriate for silicon are given in Table 15.71.

8.6.3 Brooks–Herring model

$$\mu_{eh} = \frac{c_1\left(\frac{T}{T_0}\right)^{3/2}}{\sqrt{np}}\frac{1}{\phi(\eta_0)} \tag{15.169}$$

$$\text{with}\quad \phi(\eta_0) = \ln(1+\eta_0)-\frac{\eta_0}{1+\eta_0} \quad\text{and}\quad \eta_0(T) = \frac{c_2}{N_cF_{-1/2}\left(\frac{n}{N_c}\right)+N_vF_{-1/2}\left(\frac{p}{N_v}\right)}\left(\frac{T}{T_0}\right)^2 \tag{15.170}$$

where T denotes the lattice temperature, T_0 = 300 K, n and p are the electron and hole densities, respectively, and $F_{-1/2}(y)$ is the derivative of the Fermi integral. Table 15.72 lists the silicon default values for c_1 and c_2.

8.6.4 Physical model parameters

Parameters for the carrier–carrier scattering models are accessible in the parameter file in the section:

```
CarrierCarrierScattering:{ ... }
```

Table 15.71 Conwell–Weisskopf model: Default parameters

Symbol	Parameter name	Value	Unit
D	`D`	1.04×10^{21}	1/(cmVs)
F	`F`	7.452×10^{13}	cm^{-2}

Table 15.72 Brooks-Herring model: Default parameters

Symbol	Parameter name	Value	Unit
c_1	`c1`	1.56×10^{21}	1/(cmVs)
c_2	`c2`	7.63×10^{19}	cm^{-3}

Figure 15.32 shows a comparison between the Conwell–Weisskopf and Brooks–Herring mobilities for the special case of n = p.

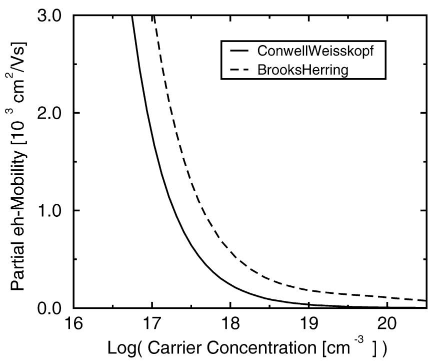


Figure 15.32 Comparison of e-h-mobility models for n = p

8.7 Philips unified mobility model

The Philips unified mobility model, proposed by Klaassen [66], unifies the description of majority and minority carrier bulk mobilities. In addition to describing the temperature dependence of the mobility, the model takes into account electron–hole scattering, screening of ionized impurities by charge carriers, and clustering of impurities. Two parameter sets apply for electron mobility, which are optimized for situations where the dominant donor species is either arsenic or phosphorus. For holes, only a single parameter set appropriate for predominantly boron-doped silicon is assumed.

The Philips unified mobility model is well calibrated. Though it was initially used primarily for bipolar devices, it is widely used for MOS devices.

8.7.1 Syntax and implementation

The Philips unified mobility model is activated by specifying the `PhuMob` argument for the `Mobility` keyword:

```
Physics{ Mobility( PhuMob ...) ...}
```

The model can also be activated with one or two additional arguments:

```
Physics{ Mobility( PhuMob( [ Arsenic | Phosphorus ]
                           [ Klaassen | Meyer ] ) ...) ...}
```

The argument `Arsenic` or `Phosphorus` specifies which of the two electron parameter sets (see Table 15.74 on page 15.192) is used. These parameters reflect the different electron mobility degradation that is observed in the presence of these donor species.

The argument `Klaassen` or `Meyer` specifies which parameter sets (see Table 15.73 on page 15.191) are used for the evaluation of $G(P_i)$. DESSIS defaults to the `Klaassen` set.

NOTE The Philips unified mobility model describes mobility degradation due to both impurity scattering and carrier–carrier scattering mechanisms. Therefore, the keyword `PhuMob` must not be combined with the keywords `DopingDependence` or `CarrierCarrierScattering`. If a combination of these keywords is specified, DESSIS uses only the Philips unified mobility model.

8.7.2 Physical model description

According to the Philips unified mobility model, there are two contributions to carrier mobilities. The first, $\mu_{i,L}$, represents phonon (lattice) scattering and the second, $\mu_{i,DAeh}$, accounts for all other bulk scattering mechanisms (due to free carriers, and ionized donors and acceptors). These partial mobilities are combined to give the bulk mobility $\mu_{i,b}$ for each carrier according to Mathiessen's rule:

$$\frac{1}{\mu_{i,b}} = \frac{1}{\mu_{i,L}} + \frac{1}{\mu_{i,DAeh}} \tag{15.171}$$

In (Eq. 15.171) and all of the following model equations, the index i takes the value e for electrons and h for holes. The first contribution due to lattice scattering takes the form:

$$\mu_{i,L} = \mu_{i,\max}\left(\frac{T}{T_0}\right)^{-\theta_i} \tag{15.172}$$

where T denotes the lattice temperature and $T_0 = 300\text{ K}$.

The second contribution has the form:

$$\mu_{i,DAeh} = \mu_{i,N}\left(\frac{N_{i,sc}}{N_{i,sc,eff}}\right)\left(\frac{N_{i,ref}}{N_{i,sc}}\right)^{\alpha_i} + \mu_{i,c}\left(\frac{n+p}{N_{i,sc,eff}}\right) \tag{15.173}$$

with:

$$\mu_{i,N} = \frac{\mu_{i,\max}^2}{\mu_{i,\max} - \mu_{i,\min}}\left(\frac{T}{T_0}\right)^{3\alpha_i - 1.5} \tag{15.174}$$

$$\mu_{i,c} = \frac{\mu_{i,\max}\mu_{i,\min}}{\mu_{i,\max} - \mu_{i,\min}}\left(\frac{T}{T_0}\right)^{0.5} \tag{15.175}$$

for the electrons:

$$N_{i,sc} = N_{e,sc} = N_D^* + N_A^* + p \tag{15.176}$$

$$N_{i,sc,eff} = N_{e,sc,eff} = N_D^* + G(P_e)N_A^* + \frac{p}{F(P_e)} \tag{15.177}$$

and for holes:

$$N_{i,sc} = N_{h,sc} = N_A^* + N_D^* + n \tag{15.178}$$

$$N_{i,sc,eff} = N_{h,sc,eff} = N_A^* + G(P_h)N_D^* + \frac{n}{F(P_h)} \tag{15.179}$$

The electron and hole concentrations are denoted by n and p, respectively.

The effects of clustering of donors (N_D^*) and acceptors (N_A^*) at ultrahigh concentrations are described by 'clustering' functions Z_D and Z_A, which are defined as:

$$N_D^* = N_D Z_D = N_D \left[1 + \frac{1}{c_D + \left(\frac{N_{D,\mathrm{ref}}}{N_D} \right)^2} \right] \qquad (15.180)$$

$$N_A^* = N_A Z_A = N_A \left[1 + \frac{1}{c_A + \left(\frac{N_{A,\mathrm{ref}}}{N_A} \right)^2} \right] \qquad (15.181)$$

The analytic functions G(Pi) and F(Pi) in (Eq. 15.177) and (Eq. 15.179) describe minority impurity and electron–hole scattering.

They are given by:

$$F(P_i) = \frac{0.7643 P_i^{0.6478} + 2.2999 + 6.5502 \left(\frac{m^*_i}{m^*_j} \right)}{P_i^{0.6478} + 2.3670 - 0.8552 \left(\frac{m^*_i}{m^*_j} \right)} \qquad (15.182)$$

and:

$$G(P_i) = 1 - \frac{a_g}{\left[b_g + P_i \left(\frac{m_0}{m^*_i} \frac{T}{T_0} \right)^{\alpha_g} \right]^{\beta_g}} + \frac{c_g}{\left[P_i \left(\frac{m_0}{m^*_i} \frac{T}{T_0} \right)^{\alpha'_g} \right]^{\gamma_g}} \qquad (15.183)$$

where m_0 is the free carrier mass and m^*_i denotes a fit parameter (which is related to the effective carrier mass). m^*_j denotes the corresponding fit parameter for holes if i = e and for electrons if i = h.

8.7.3 Screening parameter

The screening parameter P_i is given by a weighted harmonic mean of the Brooks–Herring approach and Conwell–Weisskopf approach:

$$P_i = \left[\frac{f_{CW}}{3.97 \times 10^{13} N_{i,sc}^{-2/3}} + \frac{f_{BH}}{\frac{1.36 \times 10^{20}}{n + p} \left(\frac{m^*_i}{m_0} \right)} \right]^{-1} \left(\frac{T}{T_0} \right)^2 \qquad (15.184)$$

If the `Klaassen` parameter set is selected, the evaluation of $G(P_i)$ depends on the value of the screening parameter P_i. For values of $P_i < P_{i,min}$, $G(P_{i,min})$ is used instead of $G(P_i)$, where $P_{i,min}$ is the value at which $G(P_i)$ reaches its minimum.

Figure 15.33 shows the behavior of the above analytic functions F and G for $m^*_e = m_0$ and $m^*_h = 1.258\ m_0$.

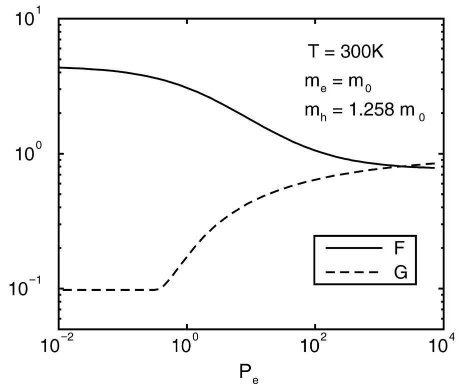


Figure 15.33 G and F as functions of screening parameter P_e for electrons

If the `Meyer` parameter set is selected, (Eq. 15.183) is used independently of the value of P_i.

8.7.4 Physical model parameters

DESSIS supports two sets of fixed values for the parameters a_g, b_g, c_g, α_g, α'_g, β_g and γ_g in (Eq. 15.183). Either set is selected by specifying either the keyword `Klaassen` or `Meyer` (see Section 8.7.1 on page 15.188). The corresponding values are given in Table 15.73. DESSIS defaults to the `Klaassen` set.

Table 15.73 Philips unified mobility model parameters: Klaassen versus Meyer

Symbol	Klaassen	Meyer	Unit
a_g	0.89233	4.41804	1
b_g	0.41372	39.9014	1
c_g	0.005978	0.52896	1
α_g	0.28227	0.0001	1
α'_g	0.72169	1.595787	1
β_g	0.19778	0.38297	1
γ_g	1.80618	0.25948	1

Other parameters for the Philips unified mobility model are accessible in the parameter file section:

```
PhuMob:{ ... }
```

Table 15.74 and Table 15.75 on page 15.192 list the silicon defaults for all the other parameters. DESSIS supports two sets of parameters for electron mobility, which are optimized for situations where the dominant donor species in the silicon is either arsenic or phosphorus. The parameter set corresponding to arsenic doping is the default. For holes, only a single parameter set appropriate for predominantly boron-doped silicon is assumed.

Table 15.74 Philips unified mobility model parameters (silicon), set 1

Symbol	Parameter name	Electrons (arsenic)	Electrons (phosphorus)	Holes (boron)	Unit
μ_{max}	`mumax_*`	1417	1414	470.5	cm^2/Vs
μ_{min}	`mumin_*`	52.2	68.5	44.9	cm^2/Vs
θ	`theta_*`	2.285	2.285	2.247	1
$N_{\{e/h\},ref}$	`n_ref_*`	9.68×10^{16}	9.2×10^{16}	2.23×10^{17}	cm^{-3}
α	`alpha_*`	0.68	0.711	0.719	1

Table 15.75 Philips unified mobility model parameters (silicon), set 2

Symbol	Parameter name	Donor (*=D)	Acceptor (*=A)	Unit
$N_{\{D/A\},ref}$	`nref_*`	4×10^{20}	7.2×10^{20}	cm^{-3}
c	`cref_*`	0.21	0.5	1

The Philips unified mobility model uses four fit parameters: the weight factors f_{CW} and f_{BH}, and the 'effective masses' m^*_e and m^*_h. The optimal parameter set, determined by accurate fitting to experimental data [66] is shown in Table 15.76.

Table 15.76 Philips unified mobility model parameters (silicon), set 3

Symbol	Parameter name	Value	Unit
m^*_e/m_0	`me_over_m0`	1	1
m^*_h/m_0	`mh_over_m0`	1.258	1
f_{CW}	`f_CW`	2.459	1
f_{BH}	`f_BH`	3.828	1

Figure 15.34 shows the majority mobility curves calculated with the model for both electrons and holes as a function of acceptor and donor concentrations.

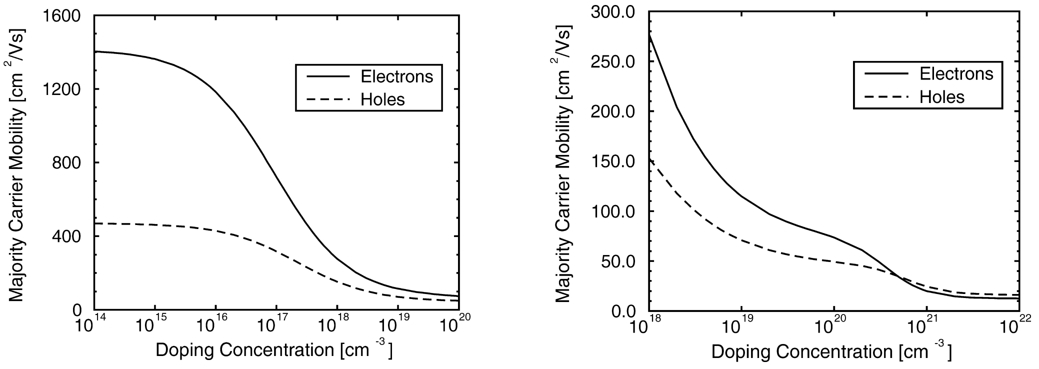


Figure 15.34 Doping dependence of majority carrier mobility in silicon using Philips unified mobility model

Figure 15.35 compares the majority and minority mobilities of electrons and holes. The minority carrier mobility is higher, in accordance with a variety of experimental data.

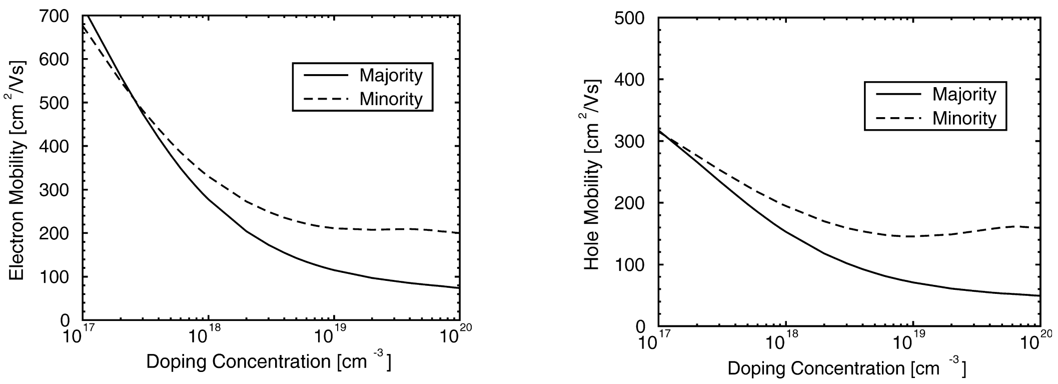


Figure 15.35 Comparison between carrier majority and minority mobilities in silicon using Philips unified mobility model

8.8 High field saturation

In high electric fields, the carrier drift velocity is no longer proportional to the electric field strength, instead, the velocity saturates to a finite speed v_{sat}. DESSIS supports different models for the description of this effect:

- The Canali model is available in different versions (one for drift-diffusion and thermodynamic, and one for hydrodynamic simulations).
- The transferred electron model is available for simulation of GaAs and related materials. It also supports a drift-diffusion/thermodynamic and a hydrodynamic version.
- The basic model and the Meinerzhagen–Engl model both require hydrodynamic simulations.

8.8.1 Syntax

The high field saturation models are activated by specifying the `HighFieldSaturation` argument for the `Mobility` keyword:

```
Physics{ Mobility( HighFieldSaturation ( <arguments> ...) ...}
```

The models can be selected for the electrons and holes independently:

```
Physics{ Mobility( [ eHighFieldSaturation ( <electron arguments> ) ]
                   [ hHighFieldSaturation ( <hole arguments> ) ] ...}
```

8.8.2 Canali model

The Canali model [67] originates from the Caughey–Thomas formula [63], but has temperature-dependent parameters, which were fitted up to 430 K by Canali et al. [67]:

$$\mu(F) = \frac{\mu_{low}}{\left[1 + \left(\frac{\mu_{low}F}{v_{sat}}\right)^{\beta}\right]^{1/\beta}} \tag{15.185}$$

where μ_{low} denotes the low field mobility. Its definition depends on which of the previously described mobility models have been activated (see Section 8.3 on page 15.176 to Section 8.7 on page 15.188). The exponent β is temperature dependent according to:

$$\beta = \beta_0 \left(\frac{T}{T_0}\right)^{\beta_{exp}} \tag{15.186}$$

where T denotes the lattice temperature and $T_0 = 300$ K. Details about the saturation velocity v_{sat} and driving field F are discussed in Section 8.8.4 on page 15.195 and Section 8.8.5 on page 15.196. All other parameters are accessible in the parameter file section:

```
HighFieldDependence:{ ...
```

The silicon default values are listed in Table 15.77 and the computed, velocity field curves at various temperatures are plotted in Figure 15.36.

Table 15.77 Canali model parameters (default values for silicon)

Symbol	Parameter name	Electrons	Holes	Unit
β_0	`beta0`	1.109	1.213	1
β_{exp}	`betaexp`	0.66	0.17	1

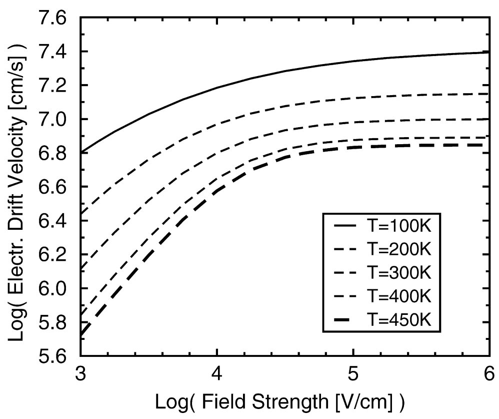


Figure 15.36 Temperature dependence of electron drift velocity according to model after Canali et al. [67]

8.8.3 Transferred electron model

For materials such as GaAs and other materials with a similar band structure, a negative differential mobility can be observed for high driving fields. This effect is caused by a transfer of electrons, heated in the high electric field, into a energetically higher side valley with a much larger effective mass. DESSIS includes a transferred electron model for the description of this effect, as given by:

$$\mu = \frac{\mu_{low} + \left(\frac{v_{sat}}{F}\right)\left(\frac{F}{E_0}\right)^4}{1 + \left(\frac{F}{E_0}\right)^4} \tag{15.187}$$

Details of the saturation velocity v_{sat} and the driving field F are discussed in Section 8.8.4 and Section 8.8.5 on page 15.196 The reference field strength E_0 can be set in the parameter file section:

```
HighFieldDependence:{ ... }
```

The `HighFieldDependence` section of the parameter file also includes a variable $K_{smooth,}$ which is equal to 1 by default. If $K_{smooth}> 1$, a smoothing algorithm is applied to the formula for mobility in the driving force interval $F_{vmax} < F < K_{smooth}F_{vmax}$, where F_{vmax} is the field strength at which the velocity is at its maximum, $v_{max} = \mu F_{vmax}$.

In this interval, (Eq. 15.187) is replaced by a polynomial that produces the same values and derivatives at the points F_{vmax} and K_{smooth} F_{vmax}. It is sometimes numerically advantageous to set $K_{smooth} \sim 20$.

Table 15.78 Transferred electron model: Default parameters

Symbol	Parameter name	Electrons	Holes	Unit
E_0	`E0_TrEf`	4.0×10^3	4.0×10^3	1
K_{smooth}	`Ksmooth_TrEf`	1	1	1

8.8.4 Velocity saturation models

DESSIS supports two velocity saturation models. `Model 1` is part of the Canali model and is given by:

$$v_{sat} = v_{sat,0}\left(\frac{T_0}{T}\right)^{v_{sat,exp}} \tag{15.188}$$

where T denotes the lattice temperature and $T_0 = 300$ K. This model is recommended for silicon. Table 15.79 lists the silicon default values.

Table 15.79 Default velocity saturation (Model 1) parameters (for silicon)

Symbol	Parameter	Electrons	Holes	Unit
$v_{sat,0}$	`vsat0`	1.07×10^7	8.37×10^6	cm/s
$v_{sat,exp}$	`vsatexp`	0.87	0.52	1

`Model 2` is recommended for GaAs, and using `Model 2`, v_{sat} is given by:

$$v_{sat} = A_{vsat} - B_{vsat}\left(\frac{T}{T_0}\right) \quad \text{for} \quad v_{sat} > v_{sat,min} \tag{15.189}$$

Otherwise, $v_{sat} = v_{sat,min}$. The parameters of `Model 1` and `Model 2` are accessible in the `HighFieldDependence` section of the parameter file.

Selecting models

The variable `Vsat_formula` in the DESSIS parameter file controls the selection of the velocity saturation models. If `Vsat_formula` is set to 1, `Model 1` is selected. If `Vsat_formula` is set to 2, `Model 2` is selected. The default value of `Vsat_formula` depends on the semiconductor material, for example, for silicon the default is 1; for GaAs, it is 2.

Table 15.80 Default velocity saturation (Model 2) parameters

Symbol	Parameter	Electrons	Holes	Unit
A_{vsat}	`A_vsat`	1.07×10^{7}	8.37×10^{6}	cm/s
B_{vsat}	`B_vsat`	3.6×10^{6}	3.6×10^{6}	cm/s
$v_{sat,min}$	`vsat_min`	5.0×10^{5}	5.0×10^{5}	cm/s

8.8.5 Driving force models

DESSIS supports two different models for the driving force F. `Model 1` sets F to be the component of the electric field parallel to the carrier current:

$$F_c = \vec{E} \bullet \left(\frac{\vec{j}_c}{|j_c|} \right) \tag{15.190}$$

where $\vec{E}$ denotes the electric field vector, and $\vec{j}$ denotes the electron (c=e) or hole (c=h) current vector. According to `Model 2`, the driving field F is given by the gradient of the Fermi potential φ_c:

$$F = |\nabla\varphi_c| \tag{15.191}$$

NOTE Usually, both models give the same or very similar results. However, numerically, one model may prove to be more stable. For example, in regions with very low current levels, the evaluation of the parallel electric field can be numerically problematic.

8.8.6 Syntax and implementation for drift-diffusion and thermodynamic simulations

For drift-diffusion and thermodynamic simulations, the high field saturation models are activated by specifying the `HighFieldSaturation` argument for the `Mobility` keyword:

```
Physics{Mobility(HighFieldSaturation ...) ...}
```

The two driving force models are selected by the additional argument:

```
Physics{ Mobility( HighFieldSaturation( [ GradQuasiFermi | Eparallel ] ) ...) ...}
```

By default, DESSIS uses `GradQuasiFermi` and the Canali model. The transferred electron model is activated by the additional argument:

```
Physics{ Mobility( HighFieldSaturation( [ GradQuasiFermi | Eparallel]
                 TransferredElectronEffect ) ...) ...}
```

8.8.7 Hydrodynamic Canali model

DESSIS supports a generalized version of the Canali model (see Section 8.8.2 on page 15.193), which can be used for hydrodynamic simulations. In this model, the driving force F is expressed is terms of the carrier thermal energy w_c (see (Eq. 15.55) and (Eq. 15.56)).

In a homogeneous and stationary situation, the hydrodynamic equations (Eq. 15.28), (Eq. 15.29), (Eq. 15.43), and (Eq. 15.44) have the simple solution:

$$F_c = \sqrt{\frac{w_c - w_0}{\tau_{e,c} q \mu}} \tag{15.192}$$

where $w_c = 3k_BT_c/2$ is the average carrier thermal energy, $w_0 = 3k_BT_L/2$ gives the equilibrium thermal energy, $\tau_{e,c}$ is the energy relaxation time, T_c denotes the carrier temperature, and T_L denotes the lattice temperature. The index c is e for electrons and h for holes. Substituting F_c into the Canali model (see (Eq. 15.185)) and solving for μ yields the hydrodynamic Canali model:

$$\mu = \frac{\mu_{\text{low}}}{\left[\sqrt{1 + \alpha^2 (w_c - w_0)^\beta} + \alpha (w_c - w_0)^{\beta/2}\right]^{2/\beta}} \tag{15.193}$$

where the parameter α is given by:

$$\alpha = \frac{1}{2}\left(\frac{\mu_{\text{low}}}{q\tau_{e,c} v_{sat}^2}\right)^{\beta/2} \tag{15.194}$$

The saturation velocity v_{sat} and the exponent β are determined in the same way as for the standard Canali model discussed in Section 8.8.2. For average carrier energies w_c less than the thermal energy w_o, the mobility is set to the low field value $\mu = \mu_{\text{low}}$.

In its given form, the hydrodynamic Canali model has a discontinuous derivative at w_c=w_0, which can lead to numeric problems. DESSIS, therefore, applies a smoothing algorithm to the kink. In the carrier temperature region $T_L < T_c < (1 + K_{dT})T_L$, this algorithm creates a smooth transition between the low field mobility μ_{low} and the mobility given in (Eq. 15.193). T_L denotes the lattice temperature and T_c denotes the carrier temperature. The parameter K_{dT} defaults to 0.2 and can be accessed in the parameter file section:

```
HighFieldDependence:{ ... }
```

8.8.8 Hydrodynamic transferred electron model

Similar to the hydrodynamic Canali model, the hydrodynamic version of the transferred electron model follows from the standard model, by replacing the driving force F according to (Eq. 15.192) and solving the resulting equation for μ.

8.8.9 A basic model

According to this very simple model, the mobility decays inversely with the carrier temperature:

$$\mu = \mu_{\mathrm{low}}\left(\frac{T_0}{T_c}\right) \tag{15.195}$$

where μ_{low} is the low field mobility, T_c is the carrier temperature, and $T_0 = 300\ \mathrm{K}$.

8.8.10 Meinerzhagen–Engl model

According to the Meinerzhagen–Engl model [68], the high field mobility degradation is given by:

$$\mu = \frac{\mu_{\mathrm{low}}}{\left[1+\left(\mu_{low}\frac{(w_c-w_0)}{q\tau_{e,c}v_{sat}^2}\right)^{\beta}\right]^{1/\beta}} \tag{15.196}$$

where $\tau_{\mathrm{e,c}}$ is the energy relaxation time. The coefficients of the saturation velocity v_{sat} (see (Eq. 15.188)) and the exponent β (see (Eq. 15.186)) are accessible the parameter file in the following section:

```
HydroHighFieldDependence:{ ... }
```

The silicon default values are given in Table 15.81.

Table 15.81 Meinerzhagen–Engl model: Default parameters

Silicon	Electrons	Holes	Unit
$v_{\mathrm{sat,0}}$	1.07×10^7	8.37×10^6	cm/s
$v_{\mathrm{sat,exp}}$	0.87	0.52	1
β_0	0.6	0.6	1
β_{exp}	0.01	0.01	1

8.8.11 Syntax and implementation for hydrodynamic simulations

For hydrodynamic simulations, the high field saturation models are activated by specifying one of the following arguments for the keyword `HighFieldSaturation`:

```
Physics { Mobility(
   HighFieldSaturation ( [CarrierTempDrive | CarrierTempDriveBasic | CarrierTempDriveME] )...)
   ...
}
```

`CarrierTempDrive` activates the hydrodynamic Canali model, `CarrierTempDriveBasic` activates the basic model and `CarrierTempDriveME` activates the Meinerzhagen–Engl model.

The hydrodynamic transferred electron model is activated by the arguments:

```
Physics { Mobility(
   HighFieldSaturation ( CarrierTempDrive TransferredElectronEffect )...)
   ...
}
```

8.9 Monte Carlo–computed mobility for strained silicon

Based on Monte Carlo simulations [174], a DESSIS parameter file `StrainedSilicon.par` is created in the library of materials (see Section 2.13.5 on page 15.92). This file contains in-plane transport parameters at 300 K for silicon under biaxial tensile strain that is present when a thin silicon film is grown on top of a relaxed `SiliconGermanium` substrate. (*In-plane* refers to charge transport that is parallel to the interface to `SiliconGermanium`, as is the case in MOSFETs.)

In the `Physics` section of the DESSIS command file, the germanium content (`XFraction`) of the `SiliconGermanium` substrate at the interface to the `StrainedSilicon` channel must be specified (this value determines the strain in the top silicon film) according to:

```
MoleFraction (RegionName=["TopLayer"] XFraction = 0.2 Grading = 0.0)
```

where the material of `TopLayer` is `StrainedSilicon`.

Band offsets and bulk mobility data are obtained from the model-solid theory of Van de Walle and descriptions of Monte Carlo simulations [174].

At present, a parameterization of the surface mobility model as a function of strain is not yet possible. The present values for the parameters `B` and `C` of the surface mobility were extracted in a 1 µm bulk MOSFET at a high effective field (where the Si control measurements of the references [175][176] were in good agreement with the universal mobility curve of Si and where the neglected effect of the SiGe substrate is minimal) of approximately 0.7 MV/cm to reproduce reported experimental data [175][176], for the typical germanium content in the `SiliconGermanium` substrate of 30%. The values for other germanium contents are given as comments.

8.10 Incomplete ionization–dependent mobility models

DESSIS supports incomplete ionization–dependent mobility models. This dependence is activated by specifying the keyword `IncompleteIonization` in `Mobility` sections. The incomplete ionization model (see Chapter 6 on page 15.161) must be activated also. The `Physics` sections for this case can be as follows:

```
Physics {
   IncompleteIonization
   Mobility( Enormal IncompleteIonization )
}
```

In this case, for all equations that contain N_A, N_D, $N_\mathrm{i} = N_\mathrm{A} + N_\mathrm{D}$, DESSIS will use N_A^-, N_D^+, $N_\mathrm{i} = N_A^- + N_D^+$.

The following mobility models depend on incomplete ionization:

- Masetti model, see (Eq. 15.148)
- Arora model, see (Eq. 15.149)
- University of Bologna bulk model, see (Eq. 15.153)–(Eq. 15.155)
- Lombardi model, see (Eq. 15.156) and (Eq. 15.159)
- University of Bologna inversion layer model, see (Eq. 15.162)–(Eq. 15.164)
- Philips unified model, see (Eq. 15.180), (Eq. 15.181)

CHAPTER 9 Generation–recombination

9.1 Shockley–Read–Hall recombination

Recombination through deep levels in the gap is usually labeled Shockley–Read–Hall (SRH) recombination. In DESSIS, the following form is implemented:

$$R_{\text{net}}^{SRH} = \frac{np - n_{i,\text{eff}}^2}{\tau_p(n+n_1) + \tau_n(p+p_1)} \tag{15.197}$$

with:

$$n_1 = n_{i,\text{eff}} e^{\frac{E_{\text{trap}}}{kT}} \tag{15.198}$$

and:

$$p_1 = n_{i,\text{eff}} e^{\frac{-E_{\text{trap}}}{kT}} \tag{15.199}$$

where E_{trap} is the difference between the defect level and intrinsic level. The variable E_{trap} is accessible in the parameter file. The silicon default value is $E_{\text{trap}} = 0$. The minority lifetimes τ_n and τ_p are modeled as a product of a doping-dependent (see Section 9.1.2 on page 15.202), field-dependent (see Section 9.2 on page 15.204 and Section 9.3 on page 15.207), and temperature-dependent (see Section 9.1.4 on page 15.203) factor:

$$\tau_c = \tau_{\text{dop}} \frac{f(T)}{1+g_c(F)}, \quad c = n, p \tag{15.200}$$

where $c = n$ or $c = p$ for holes. For an additional density dependency of the lifetimes, see Section 9.8 on page 15.213.

For simulations that use Fermi statistics (see Section 4.4 on page 15.137) or quantization (see Chapter 7 on page 15.165), (Eq. 15.197) needs to be generalized. The modified equation reads:

$$R_{\text{net}}^{SRH} = \frac{np - \gamma_n \gamma_p n_{i,\text{eff}}^2}{\tau_p(n+\gamma_n n_1) + \tau_n(p+\gamma_p p_1)} \tag{15.201}$$

where γ_n and γ_p are given by (Eq. 15.70) and (Eq. 15.71).

9.1.1 Syntax and implementation

The generation–recombination models are selected in the `Physics` section as an argument to the `Recombination` keyword:

```
Physics{ Recombination( <arguments> ) ...}
```

The SRH model is activated by specifying the `SRH` argument:

```
Physics{ Recombination( SRH ...) ...}
```

The keyword for plotting the SRH recombination rate is:

```
Plot{ ...
      SRHRecombination}
```

9.1.2 Doping dependence

The doping dependence of the SRH lifetimes is modeled in DESSIS with the Scharfetter relation:

$$\tau_{\text{dop}}(N_i) = \tau_{min} + \frac{\tau_{max} - \tau_{min}}{1 + \left(\frac{N_i}{N_{ref}}\right)^{\gamma}} \tag{15.202}$$

Such a dependence arises from experimental data [69] and the theoretical conclusion that the solubility of a fundamental, acceptor-type defect (probably a divacancy (E5) or a vacancy complex) is strongly correlated to the doping density [70]–[72]. Default values are listed in Table 15.82 on page 15.204. The Scharfetter relation is used when the argument `DopingDependence` is specified for the SRH recombination. Otherwise, $\tau = \tau_{max}$ is used.

The evaluation of the SRH lifetimes according to the Scharfetter model is activated by specifying the additional argument `DopingDependence` for the `SRH` keyword in the `Recombination` statement:

```
Physics{ Recombination( SRH( DopingDependence ... )  ...) ...}
```

A plot of electron and hole lifetimes as functions of the doping concentration is presented in Figure 15.37 on page 15.203.

9.1.3 Lifetime profiles from MDRAW

Lifetime profiles can be defined with the doping profiles and the grid, by using the device editor MDRAW (see MDRAW, Section 3.2 on page 11.25). These profiles are loaded into DESSIS by using the keyword `LifeTime` in the `File` section to specify the MDRAW output file containing the lifetime profiles, for example:

```
File{
   Grid     = "MyDev_mdr.grd"
   Doping   = "MyDev_mdr.dat"
   LifeTime = "MyDev_mdr.dat"
   ...
}
```

For each grid point, the values defined by the lifetime profile are used as τ_{max} in (Eq. 15.202).

The lifetime data can be in a separate `.dat` file from the doping, but must correspond to the same grid.

NOTE Recombination lifetimes depend strongly on processing conditions and may vary within a given device. Therefore, SRH lifetimes can be regarded as fitting parameters.

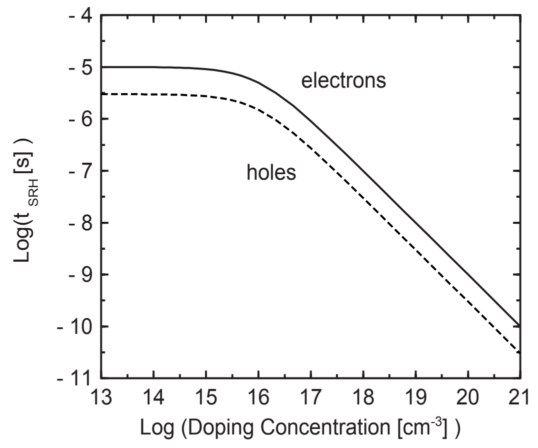


Figure 15.37 Doping dependence of SRH lifetimes according to Scharfetter relation (Eq. 15.202)

9.1.4 Temperature dependence

To date, there is no consensus on the temperature dependence of the SRH lifetimes. This appears to originate from a different understanding of *lifetime*. From measurements of the *recombination lifetime* [74]–[76]:

$$\tau = \delta n / R \tag{15.203}$$

in power devices (δn is the excess carrier density under neutral conditions, $\delta n = \delta p$), it was concluded that the lifetime increases with rising temperature. Such a dependence was modeled either by a power law [74][75]:

$$\tau(T) = \tau_0 \left(\frac{T}{300}\right)^{\alpha} \tag{15.204}$$

or an exponential expression of the form [76]:

$$\tau(T) = \tau_0 e^{C\left(\frac{T}{300} - 1\right)} \tag{15.205}$$

A calculation using the low-temperature approximation of multiphonon theory [73] gives:

$$\tau_{\mathrm{SRH}}(T) = \tau_{\mathrm{SRH}}(300K) \cdot f(T) \text{ with } f(T) = \left(\frac{T}{300}\right)^{T_\alpha} \tag{15.206}$$

with $T_\alpha = -3/2$, which is the expected decrease of minority carrier lifetimes with rising temperature. Since the temperature behavior strongly depends on the nature of the recombination centers, there is no universal law $\tau_{\mathrm{SRH}}(T)$.

In DESSIS, the power law model, (Eq. 15.204), can be activated with the keyword `TempDependence` in the `SRH` statement:

```
Physics{ Recombination( SRH( TempDependence ...) ...}
```

Additionally, DESSIS supports an exponential model for $f(T)$:

$$f(T) = e^{C\left(\frac{T}{300} - 1\right)} \qquad (15.207)$$

This model is activated with the keyword `ExpTempDependence`:

```
Physics{ Recombination( SRH( ExpTempDependence ...) ...}
```

9.1.5 SRH model parameters

All the parameters of the doping- and temperature-dependent SRH recombination models are accessible in the parameter file section:

```
Scharfetter{ ...}
```

Table 15.82 Default parameters for doping- and temperature-dependent SRH lifetime

Symbol	Parameter name	Electrons	Holes	Unit
τ_{min}	`taumin`	0	0	s
τ_{max}	`taumax`	$1\text{x}10^{-5}$	$3\text{x}10^{-6}$	s
N_{ref}	`Nref`	$1\text{x}10^{16}$	$1\text{x}10^{16}$	cm^{-3}
γ	`gamma`	1	1	1
T_α	`Talpha`	–1.5	–1.5	1
C	`Tcoeff`	2.55	2.55	1
E_{trap}	`Etrap`	0	0	eV

9.2 Trap-assisted tunneling/SRH

Trap-assisted tunneling (also known as defect-assisted tunneling or field-enhanced recombination) results in a reduction of SRH recombination lifetimes in regions of strong electric fields. It must not be neglected if the electric field exceeds a value of approximately 3×10^5 V/cm in certain regions of the device. For example, the I–V characteristics of reverse biased pn-junctions are extremely sensitive to defect-assisted tunneling, which causes electron–hole pair generation before band-to-band tunneling or avalanche generation sets in. Therefore, it is recommended that this model is included in the simulation of drain reverse leakage and substrate currents in MOS transistors.

9.2.1 Syntax and implementation

The local field–dependence of the SRH lifetimes is activated by the keyword `Tunneling`:

```
Physics{ Recombination( SRH ( Tunneling ... )  ...) ...}
```

NOTE In some situations, the inclusion of defect-assisted tunneling may lead to convergence problems. In such cases, it is recommended that the keyword `NoSRHperPotential` is specified in the `Math` section:

```
Math{ NoSRHperPotential ...}
```

This causes DESSIS to exclude derivatives of g(F) with respect to the potential from the Jacobian matrix.

9.2.2 Model description

The field dependence of the recombination rate is taken into account by the field enhancement factors:

$$[1+g(F)]^{-1} \tag{15.208}$$

of the SRH lifetimes [73] where F denotes the field strength (see (Eq. 15.200)). Furthermore, the density n in (Eq. 15.197) is replaced by:

$$\tilde{n} = n\exp\left(-\frac{\left|\nabla E_{\mathrm{F}_n}\right|(E_t-E_0)}{k_{\mathrm{B}}TF}\right) \tag{15.209}$$

with E_0 and E_t according to (Eq. 15.211) and (Eq. 15.212). p is replaced by an analogous expression. In the case of electrons, $g(F)$ has the form:

$$g_n(F) = \left(1+\frac{(\hbar\Theta)^{3/2}\sqrt{E_t-E_0}}{E_0\hbar\omega_0}\right)^{-\frac{1}{2}}\frac{(\hbar\Theta)^{3/4}(E_t-E_0)^{1/4}}{2\sqrt{E_tE_0}}\left(\frac{\hbar\Theta}{k_{\mathrm{B}}T}\right)^{\frac{3}{2}}$$
$$\times e^{\left(-\frac{E_t-E_0}{\hbar\omega_0}+\frac{\hbar\omega_0-k_{\mathrm{B}}T}{2\hbar\omega_0}+\frac{E_t+(k_{\mathrm{B}}T)/2}{\hbar\omega_0}\ln(E_t/\varepsilon_R)-\frac{E_0}{\hbar\omega_0}\ln(E_0/\varepsilon_R)\right)}$$
$$\times e^{\left(\frac{E_t-E_0}{k_{\mathrm{B}}T}\right)} e^{\left(-\frac{4}{3}\left(\frac{E_t-E_0}{\hbar\Theta}\right)^{\frac{3}{2}}\right)} \tag{15.210}$$

where E$_0$ denotes the energy of an optimum horizontal transition path, which depends on field strength and temperature in the following way:

$$E_0 = 2\sqrt{\varepsilon_F}[\sqrt{\varepsilon_F+E_t+\varepsilon_R}-\sqrt{\varepsilon_F}]-\varepsilon_R,\ \varepsilon_F = \frac{(2\varepsilon_R k_{\mathrm{B}}T)^2}{(\hbar\Theta)^3} \tag{15.211}$$

In this expression, $\varepsilon_R = S\hbar\omega_0$ is the lattice relaxation energy, S is the Huang–Rhys factor, $\hbar\omega_0$ is the effective phonon energy, E$_t$ is the energy level of the recombination center (thermal depth), and $\Theta = (e^2F^2/2\hbar m_{\Theta,\mathrm{e}})^{1/3}$ is the electro-optical frequency. The mass $m_{\Theta,\mathrm{e}}$ is the electron tunneling mass in the field direction and is given in the parameter file. The expression for holes follows from (Eq. 15.210) by replacing $m_{\Theta,\mathrm{e}}$ with $m_{\Theta,\mathrm{h}}$ and E_t with E_g-E_t.

For electrons, E_t is related to the defect level E_{trap} of (Eq. 15.198) and (Eq. 15.199) by:

$$E_t = \frac{1}{2}E_{\mathrm{g,eff}} + \frac{3}{4}k_{\mathrm{B}}T\ln\left(\frac{m_c}{m_v}\right) - E_{\mathrm{trap}} - (32R_{\mathrm{C}}\hbar^3\Theta^3)^{1/4} \qquad (15.212)$$

where m_v and m_c are taken from (Eq. 15.111) and (Eq. 15.116), respectively, and the effective Rydberg constant R_{C} is:

$$R_{\mathrm{C}} = m_c\left(\frac{Z^2}{\varepsilon^2}\right)Ry \qquad (15.213)$$

where Ry is the Rydberg energy (13.606 eV), ε is the relative dielectric constant, and Z is a fit parameter. For holes, E_t is given by:

$$E_t = \frac{1}{2}E_{\mathrm{g,eff}} - \frac{3}{4}k_{\mathrm{B}}T\ln\left(\frac{m_c}{m_v}\right) + E_{\mathrm{trap}} - (32R_{\mathrm{V}}\hbar^3\Theta^3)^{1/4} \qquad (15.214)$$

Note that E_{trap} is measured from the intrinsic level and not from mid gap. The zero-field lifetime τ_{SRH} is defined by (Eq. 15.202).

Figure 15.38 shows that the lifetime reduction starts at approximately 3×10^5 V/cm.

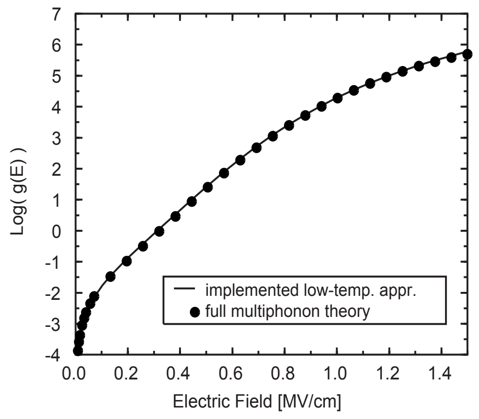


Figure 15.38 Field enhancement factor g_n for SRH lifetime calculated with default parameters and T = 300 K

The solid curve represents (Eq. 15.210), which is based on the low-temperature approximation of multiphonon theory. For comparison, the dots show the exact results.

9.2.3 Model parameters

The parameters for the trap-assisted SRH lifetimes are accessible in the parameter file in the section:

```
TrapAssistedTunneling : { ... }
```

The default parameters implemented in DESSIS are related to the gold acceptor level: $E_{\mathrm{trap}} = 0$ eV, $S = 3.5$, and $\hbar\omega_0 = 0.068$ eV.

9.3 Hurkx trap-assisted tunneling model

The commonly used Hurkx trap-assisted tunneling model is implemented in DESSIS and is used to:

- Reduce SRH lifetimes in regions of high electric field.
- Increase the cross section of traps in the trap equations.

9.3.1 Syntax and implementation

To activate the Hurkx tunneling model for SRH recombination, specify the following keywords in `Physics` section of the input file:

```
Physics{ ...
   Recombination( SRH(tunneling(Hurkx)) )
...}
```

Alternatively, to apply the model to trap equations, the same keywords must be included in the trap definition for a trap distribution:

```
Physics{ ...
   Traps( (...) (... Tunneling(Hurkx)) (...) )
...}
```

There is another possibility to switch on the model for trap equations (see Chapter 10 on page 15.225) if the user needs to apply the model for all the trap distributions of a specified region. This is achieved by changing `XsecFormula` for the cross section calculation in the `Traps` section of the parameter file:

```
Traps{
   *   XsecFormula=1: Xsec(F) = Xsec
   *   XsecFormula=2: Xsec(F) = Xsec*(1+a1*(F/F0)^p1+a2*(F/F0)^p2)^p0
   *   XsecFormula=3: Xsec(F) = Xsec*(1+Gt)
   XsecFormula = 3, 3
}
```

9.3.2 Model description

The following equations apply to electrons and holes. The lifetimes and capture cross sections become functions of the trap-assisted tunneling factor Γ_{tat}:

$$\tau = \tau_0/(1+\Gamma_{tat}), \quad \sigma = \sigma_0 \cdot (1+\Gamma_{tat}) \tag{15.215}$$

where Γ_{tat} is given by:

$$\Gamma_{tat} = \int_0^{\tilde{E}_n} \exp\left[u - \frac{2}{3}\frac{\sqrt{u^3}}{\tilde{E}}\right] du \tag{15.216}$$

with the approximate solutions:

$$\Gamma_{tat} \approx \begin{cases} \sqrt{\pi}\tilde{E}\cdot\exp\left[\frac{1}{3}\tilde{E}^2\right]\cdot\left(2-erfc\left[\frac{1}{2}\left(\frac{\tilde{E}_n}{\tilde{E}}-\tilde{E}\right)\right]\right), \tilde{E}\le\sqrt{\tilde{E}_n} \\ \sqrt{\pi\tilde{E}}\cdot\tilde{E}_n^{\frac{1}{4}}\exp\left[-\tilde{E}_n+\tilde{E}\sqrt{\tilde{E}_n}+\frac{1}{3}\tilde{E}\sqrt{\tilde{E}_n}^3\right]erfc\left[\tilde{E}_n^{\frac{1}{4}}\sqrt{\tilde{E}}-\tilde{E}_n^{\frac{3}{4}}/\sqrt{\tilde{E}}\right], \tilde{E}>\sqrt{\tilde{E}_n} \end{cases} \tag{15.217}$$

where $\tilde{E}$ and $\tilde{E}_n$ are respectively defined as:

$$\tilde{E} = \frac{E}{E_0}\text{, where } E_0 = \frac{\sqrt{8m_0m_t(k_BT)^3}}{qh} \tag{15.218}$$

$$\tilde{E}_n = \frac{E_n}{k_BT} = \begin{cases} 0, k_BT\ln\frac{n}{n_i} > 0.5E_g \\ \frac{0.5E_g}{k_BT}-\ln\frac{n}{n_i}, E_{trap}\le k_BT\ln\frac{n}{n_i}\le 0.5E_g \\ \frac{0.5E_g}{k_BT}-E_{trap}, E_{trap} > k_BT\ln\frac{n}{n_i} \end{cases} \tag{15.219}$$

where m_t is the carrier tunneling mass and E_{trap} is an energy of trap level that is taken from SRH recombination if the model is applied to the lifetimes (τ) or from trap equations if it is applied to cross sections (σ).

When quantization is active (see Chapter 7 on page 15.165), the classical density:

$$n_{\text{cl}} = n\exp\left(\frac{\Lambda}{k_{\text{B}}T}\right) \tag{15.220}$$

rather than the true density n enters (Eq. 15.219).

9.3.3 Model parameters

The model has only one parameter, m_t, the carrier tunneling mass, which can be specified in the DESSIS parameter file for electrons and holes as follows:

```
HurkxTrapAssistedTunneling{
   mt = <value>, <value>
}
```

9.4 Surface SRH recombination

The surface SRH recombination model can be activated at the interface between two different materials or two different regions (see Section 2.5.5 on page 15.49).

At interfaces, an additional formula is used that is structurally equivalent to the bulk expression of the SRH generation–recombination:

$$R^{SRH}_{\text{surf, net}} = \frac{np - n^2_{i,\text{eff}}}{(n+n_1)/s_p + (p+p_1)/s_n} \tag{15.221}$$

with:

$$n_1 = n_{i,\text{eff}} e^{\frac{E_{\text{trap}}}{kT}} \text{ and } p_1 = n_{i,\text{eff}} e^{\frac{-E_{\text{trap}}}{kT}} \tag{15.222}$$

For Fermi statistics and quantization, the equations are modified in the same manner as for bulk SRH recombination (see (Eq. 15.201)).

The recombination velocities of otherwise identically prepared surfaces depend, in general, on the concentration of dopants at the surface [77]–[79]. Particularly, in cases where the doping concentration varies along an interface, it is desirable to include such a doping dependence. DESSIS models doping dependence of surface recombination velocities according to:

$$s = s_0\left[1 + s_{\text{ref}}\left(\frac{N_i}{N_{\text{ref}}}\right)^{\gamma}\right] \tag{15.223}$$

The results of Cuevas [79] indicate that for phosphorus-diffused silicon, $\gamma = 1$; while the results of King [78] seem to imply that no significant doping dependence exists for the recombination velocities of boron-diffused silicon surfaces. To activate the model, specify the option `surfaceSRH` to the `Recombination` keyword in the `Physics` section for the respective interface. To plot the surface recombination, specify `SurfaceRecombination` in the `Plot` section. The parameters E_{trap}, s_{ref}, N_{ref}, and γ are accessible in the parameter file section:

```
SurfaceRecombination * surface SRH recombination: {...}
```

The corresponding values for silicon are given in Table 15.83.

Table 15.83 Surface SRH parameters

Symbol	Parameter name	Electrons	Holes	Unit
S_0	`S0`	1×10^3	1×10^3	cm/s
S_{ref}	`Sref`	1×10^3		1
N_{ref}	`Nref`	1×10^{16}		cm^{-3}
γ	`gamma`	1		1
E_{Trap}	`Etrap`	0		eV

The doping dependence of the recombination velocity can be suppressed by setting S_{ref} to zero.

9.5 Coupled defect level (CDL) recombination

The steady state recombination rate for two coupled defect levels generalizes the familiar single-level SRH formula. An important feature of the model is a possibly increased field effect that may lead to large excess currents. The model is discussed in the literature [80].

9.5.1 Syntax and implementation

The CDL recombination can be switched on using the keyword `CDL` in the `Physics` section of the input file:

```
Physics{ Recombination( CDL ...) ...}
```

The contributions R_1 and R_2 in (Eq. 15.226) can be plotted by using the keywords `CDL1` and `CDL2` in the `Plot` section. For the net rate and coupling term, $R - R_1 - R_2$, the keywords `CDL` and `CDL3` must be specified.

9.5.2 Model description

The notation of the model parameters is illustrated in Figure 15.39.

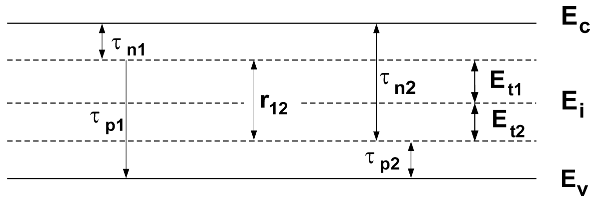


Figure 15.39 Notation for CDL recombination including all capture and emission processes

The CDL recombination rate is given by:

$$R = R_1 + R_2 + \left(\sqrt{R_{12}^2 - S_{12}} - R_{12}\right) \times \frac{\tau_{n1}\tau_{p2}(n+n_2)(p+p_1) - \tau_{n2}\tau_{p1}(n+n_1)(p+p_2)}{r_1 r_2} \tag{15.224}$$

with:

$$r_j = \tau_{nj}(p+p_j) + \tau_{pj}(n+n_j) \tag{15.225}$$

$$R_j = \frac{np - n_{i,eff}^2}{r_j}, \qquad j = 1, 2 \tag{15.226}$$

$$R_{12} = \frac{r_1 r_2}{2r_{12}\tau_{n1}\tau_{n2}\tau_{p1}\tau_{p2}(1-\varepsilon)} + \frac{\tau_{n1}(p+p_1) + \tau_{p2}(n+n_2)}{2\tau_{n1}\tau_{p2}(1-\varepsilon)} + \frac{\varepsilon[\tau_{n2}(p+p_2) + \tau_{p1}(n+n_1)]}{2\tau_{n2}\tau_{p1}(1-\varepsilon)} \tag{15.227}$$

$$S_{12} = \frac{1}{\tau_{n1}\tau_{p2}(1-\varepsilon)}\left(1 - \frac{\tau_{n1}\tau_{p2}}{\tau_{n2}\tau_{p1}}\varepsilon\right)(np - n_{i,eff}^2) \tag{15.228}$$

$$\varepsilon = \exp\left(-\frac{|E_{t2} - E_{t1}|}{k_B T}\right) \tag{15.229}$$

where the terms n, p, $n_{i,eff}$, τ_{ni}, and τ_{pi} denote the electron density, hole density, effective intrinsic density, electron lifetime of defect level i, and hole lifetime of level i, respectively. The coupling parameter between the defect levels is called r_{12} (keyword `TrapTrapRate` in the parameter file). The carrier lifetimes are calculated analogously to the carrier lifetimes in the SRH model (see Section 9.1 on page 15.201). The number of parameters is doubled compared to the SRH model, and they are changeable in the parameter file section:

```
CDL : { ... }
```

The quantities n_2 and p_2 are the corresponding quantities of n_1 and p_1 for the second defect level. They are defined analogously to (Eq. 15.198) and (Eq. 15.199).

For Fermi statistics and quantization, the equations are modified in the same manner as for SRH recombination (see (Eq. 15.201)).

9.6 Radiative recombination model

9.6.1 Syntax and implementation

The radiative recombination model is activated in the `Physics` section of the DESSIS input file by the keyword `Radiative`:

```
Physics {
   Recombination (Radiative)
}
```

It can also be switched on or off by using the notation `+Radiative` or `-Radiative`:

```
Physics (Region = "gate") {
   Recombination (+Radiative)
}

Physics (Material = "AlGaAs") {
   Recombination (-Radiative)
}
```

The value of the radiative recombination rate is plotted as follows:

```
Plot {
   RadiativeRecombination
}
```

The value of the parameter C can be changed in a section of the DESSIS parameter file:

```
RadiativeRecombination {
   C = 2.5e-10
}
```

9.6.2 Model description

The radiative (direct) recombination model expresses the recombination rate as a function of the carrier concentrations n and p, and the effective intrinsic density $n_{i,\text{eff}}$:

$$R = C \cdot (np - n_{i,\text{eff}}^2) \tag{15.230}$$

By default, DESSIS selects $C = 2 \cdot 10^{-10}$ cm^3/s for GaAs and $C = 0$ cm^3/s for other materials. For Fermi statistics and quantization, the equations are modified in the same manner than for SRH recombination (see (Eq. 15.201)).

9.7 Auger recombination

The rate of band-to-band Auger recombination R^A is given by:

$$R^A = \left(C_n n + C_p p\right)\left(np - n_{i,\,eff}^2\right) \tag{15.231}$$

with temperature-dependent Auger coefficients [81]–[83]:

$$C_n(T) = \left(A_{A,\,n} + B_{A,\,n}\left(\frac{T}{T_0}\right) + C_{A,\,n}\left(\frac{T}{T_0}\right)^2\right) \cdot \left(1 + H_n e^{-\frac{n}{N_{0,\,n}}}\right) \tag{15.232}$$

$$C_p(T) = \left(A_{A,\,p} + B_{A,\,p}\left(\frac{T}{T_0}\right) + C_{A,\,p}\left(\frac{T}{T_0}\right)^2\right) \cdot \left(1 + H_p e^{-\frac{p}{N_{0,\,p}}}\right) \tag{15.233}$$

where $T_0 = 300\text{ K}$. There is experimental evidence for a decrease of the Auger coefficients at high injection levels [83]. This effect is explained as resulting from exciton decay: at lower carrier densities, excitons, which are loosely bound electron–hole pairs, increase the probability for Auger recombination. Excitons decay at high carrier densities, resulting in a decrease of recombination. This effect is modeled by the terms (1+H exp(–n/N$_o$)) in (Eq. 15.232) and (Eq. 15.233).

Auger recombination is typically important at high carrier densities. Therefore, this injection dependence will only be seen in devices where extrinsic recombination effects are extremely low, such as high-efficiency silicon solar cells. The injection dependence of the Auger coefficient can be deactivated by setting `H` to zero in the parameter file.

For Fermi statistics and quantization, the equations are modified in the same manner as for SRH recombination (see (Eq. 15.201)). Default values for silicon are listed in Table 15.84.

Table 15.84 Default coefficients of Auger recombination model

Symbol	A_A [cm^6/s]	B_A [cm^6/s]	C_A [cm^6/s]	H (1)	N$_0$ [cm^{-3}]
Parameter name	*A*	*B*	*C*	H	N0
Electrons	0.67 x 10^{-31}	2.45 x 10^{-31}	–2.2 x 10^{-32}	3.46667	1x10^{18}
Holes	0.72 x 10^{-31}	4.50 x 10^{-33}	2.63 x 10^{-32}	8.25688	1x10^{18}

Auger recombination is activated with the argument `Auger` in the `Recombination` statement:

```
Physics{ Recombination( Auger ...) ...}
```

By default, DESSIS uses (Eq. 15.231) only if R^A is positive and replaces the value by zero if R^A is negative. To use (Eq. 15.231) for negative values (that is, to allow for Auger generation of electron-hole pairs), use the `WithGeneration` option to the `Auger` keyword:

```
Physics { Recombination( Auger(WithGeneration) ...) ...}
```

The Auger parameters are accessible in the parameter file in the section:

```
Auger : { ... }
```

9.8 Trap-assisted Auger recombination

Trap-assisted Auger recombination (TAA) is not an independent generation–recombination model, but a modification to the SRH recombination (see Section 9.1 on page 15.201) and coupled defect level (see Section 9.5 on page 15.209) models. When TAA is active, DESSIS uses the lifetimes [72]:

$$\frac{\tau_p}{1+\tau_p/\tau_p^{TAA}} \tag{15.234}$$

$$\frac{\tau_n}{1+\tau_n/\tau_n^{TAA}} \tag{15.235}$$

in place of the lifetimes τ_p and τ_n in (Eq. 15.197) and (Eq. 15.225).

The TAA lifetimes in (Eq. 15.234) depend on the carrier densities:

$$\frac{1}{\tau_n^{TAA}} = N_t\left(C_n^{\mathrm{TAA,e}}n + C_n^{\mathrm{TAA,h}}p\right) \approx c_p^{TAA}(n+p) \tag{15.236}$$

$$\frac{1}{\tau_p^{TAA}} = N_t\left(C_p^{\mathrm{TAA,e}}n + C_p^{\mathrm{TAA,h}}p\right) \approx c_n^{TAA}(n+p) \tag{15.237}$$

A reasonable order of magnitude for the TAA coefficients is $C_v^{TAA} \sim (1\times10^{-12} - 1\times10^{-11})\ \mathrm{cm^3s^{-1}}$; default values are $C_n^{TAA} = C_p^{TAA} = 1\times10^{-12}\ \mathrm{cm^3s^{-1}}$.

TAA recombination is activated by using the keyword `TrapAssistedAuger` in the `Recombination` statement in the `Physics` section of the input file (see Section 2.5.2.3 on page 15.47):

```
Physics{
   Recombination(TrapAssistedAuger ...)
   ...
}
```

The trap-assisted Auger parameters C_n^{TAA} and C_p^{TAA} are accessible in the parameter file in the section:

```
TrapAssistedAuger: { ... }
```

9.9 Avalanche generation

Electron–hole pair production due to avalanche generation (impact ionization) requires a certain threshold field strength and the possibility of acceleration, that is, wide space charge regions. If the width of a space charge region is greater than the mean free path between two ionizing impacts, charge multiplication occurs, which can cause electrical breakdown. The reciprocal of the mean free path is called the ionization coefficient α. With these coefficients for electrons and holes, the generation rate can be expressed as:

$$G^{\|} = \alpha_n n v_n + \alpha_p p v_p \tag{15.238}$$

where $v_{n,p}$ denotes the drift velocity. DESSIS implements three models of the threshold behavior of the ionization coefficients: van Overstraeten – de Man, Okuto–Crowell, and Lackner.

DESSIS allows users to select the appropriate driving force for the simulation, that is, the method used to compute the accelerating field.

9.9.1 Syntax and implementation

Avalanche generation is switched on by using the keyword `Avalanche` in the `Recombination` statement in the `Physics` section of the input file. The models are selected by using the keywords `vanOverstraeten`, `Lackner`, and `UniBo`. The default model is `vanOverstraeten`. For example:

```
Physics{
   Recombination(eAvalanche(CarrierTempDrive) hAvalanche(Okuto)...
}
```

selects a driving force derived from electron temperature for electron impact ionization process and the default driving force based on `GradQuasiFermi` with the Okuto–Crowell model for holes.

9.9.2 van Overstraeten – de Man model

This model is based on the Chynoweth law [84]:

$$\alpha(F) = \gamma a e^{-\frac{\gamma b}{F}} \tag{15.239}$$

with:

$$\gamma = \frac{\tanh\left(\frac{h\omega_{op}}{2kT_0}\right)}{\tanh\left(\frac{h\omega_{op}}{2kT}\right)} \tag{15.240}$$

The factor γ with the optical phonon energy $h\omega_{op}$ expresses the temperature dependence of the phonon gas against which carriers are accelerated. The coefficients a, b, and $h\omega_{op}$, as measured by van Overstraeten and de Man [85], are applicable over the range of fields $1.75 \times 10^5 - 6 \times 10^5$ and are listed in Table 15.85 on page 15.215.

NOTE Two sets of coefficients a and b are used for high and low ranges of electric field. The values `a(low)`, `b(low)` apply in the low field range 1.75 x 10^5 – E_0 eV and the values `a(high)`, `b(high)` apply in the high field range E_0 – 6 x 10^5 eV. For electrons, the impact ionization coefficients are by default the same in both field ranges.

The user can adjust the coefficient values in the `dessis.par` file in the section:

```
vanOverstraetendeMan * Impact Ionization:
```

Table 15.85 Coefficients for van Overstraeten – de Man model, (Eq. 15.239)

Symbol	Parameter name	Electrons	Holes	Valid range of electric field	Unit
a	`a(low)`	7.03×10^5	1.582×10^6	$1.75 \times 10^5 - E_0$	cm^{-1}
	`a(high)`	7.03×10^5	6.71×10^5	$E_0 - 6 \times 10^5$	
b	`b(low)`	1.231×10^6	2.036×10^6	$1.75 \times 10^5 - E_0$	V/cm
	`b(high)`	1.231×10^6	1.693×10^6	$E_0 - 6 \times 10^5$	
E_0	`E0`	4×10^5	4×10^5		V/cm
$h\omega_{op}$	`hbarOmega`	0.063	0.063		eV

9.9.3 Okuto–Crowell model

Okuto and Crowell [86] suggested the empirical model:

$$\alpha(F) = a \cdot \left(1 + c(T - T_0)\right) \cdot F^{\gamma} \cdot e^{-\left(\frac{b[1 + d(T - T_0)]}{F}\right)^{\delta}} \tag{15.241}$$

where T_0=300 K and the user-adjustable coefficients are listed in Table 15.86 with their default values for silicon. These values are applicable to the range of electric field 10^5–10^6 V/cm. The user can adjust these values in the following section of the DESSIS parameter file:

```
OkutoCrowell * Impact Ionization
```

Table 15.86 Coefficients for Okuto–Crowell model, (Eq. 15.241)

Symbol	Parameter name	Electrons	Holes	Unit
a	`a`	0.426	0.243	V^{-1}
b	`b`	4.81×10^5	6.53×10^5	V/cm
c	`c`	3.05×10^{-4}	5.35×10^{-4}	K^{-1}
d	`d`	6.86×10^{-4}	5.67×10^{-4}	K^{-1}
γ	`gamma`	1	1	1
δ	`delta`	2	2	1

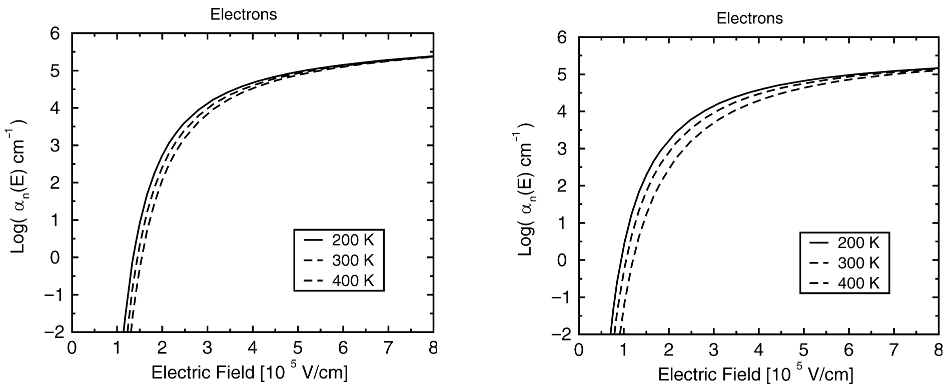


Figure 15.40 Temperature dependence of e-impact ionization rate after Okuto–Crowell model (*left*) and Lackner model (*right*)

9.9.4 Lackner model

Lackner [87] derived a pseudo-local ionization rate in the form of a modification to the Chynoweth law, assuming stationary conditions. The temperature-dependent factor γ was introduced to the original model:

$$\alpha_\nu(F) = \frac{\gamma a_\nu}{Z} e^{-\frac{\gamma b_\nu}{F}} \quad \text{where } \nu = n, p \tag{15.242}$$

with:

$$Z = 1 + \frac{\gamma b_n}{F} e^{-\frac{\gamma b_n}{F}} + \frac{\gamma b_p}{F} e^{-\frac{\gamma b_p}{F}} \tag{15.243}$$

and:

$$\gamma = \frac{\tanh\left(\frac{h\omega_{op}}{2kT_0}\right)}{\tanh\left(\frac{h\omega_{op}}{2kT}\right)} \tag{15.244}$$

The default values of the coefficients a, b, and $h\omega_{op}$ are applicable in silicon for the range of electric field 10^5–10^6 V/cm. The user can adjust these values in the parameter file in the section:

```
Lackner * Impact Ionization
```

Table 15.87 Coefficients for Lackner model, (Eq. 15.242)

Symbol	Parameter name	Electrons	Holes	Unit
a	`a`	1.316 x 10^6	1.818 x 10^6	cm^{-1}
b	`b`	1.474 x 10^6	2.036 x 10^6	V/cm
$h\omega_{op}$	`hbarOmega`	0.063	0.063	eV

9.9.5 University of Bologna impact ionization model

The University of Bologna impact ionization model was developed for an extended temperature range between 25°C and 400°C. It is based on impact ionization data generated by the Boltzmann solver HARM [149]. It covers a wide range of electric fields (50–600 kV/cm) and temperatures (300–700 K). It is calibrated against impact ionization measurements [145][148] in the whole temperature range. The model reads:

$$\alpha_{n,p}(F,T) = \frac{F}{a(T) + b(T)\exp\left[\frac{d(T)}{F + c(T)}\right]} \tag{15.245}$$

The temperature dependence of the model parameters, determined by fitting experimental data, reads (for electrons):

$$a(T) = a_0 + a_1 T^{a_2} \qquad b(T) = b_0 \qquad c(T) = c_0 + c_1 T + c_2 T^2 \qquad d(T) = d_0 + d_1 T + d_2 T^2 \tag{15.246}$$

and for holes:

$$a(T) = a_0 + a_1 T \qquad b(T) = b_0 \exp[b_1 T] \qquad c(T) = c_0 T^{c_1} \qquad d(T) = d_0 + d_1 T + d_2 T^2 \tag{15.247}$$

Table 15.88 lists the model parameters.

Table 15.88 Coefficients for University of Bologna impact ionization model

Silicon	Parameter name	Electrons	Holes	Unit
a_0	`ha0`	4.338	2.376	V
a_1	`ha1`	-2.42×10^{-12}	1.033×10^{-2}	$V/(K)^{a2}$, V/K
a_2	`ha2`	4.123	0	1
b_0	`hb0`	0.235	0.177	V
b_1	`hb1`	0	-2.178×10^{-3}	1/K
c_0	`hc0`	1.68×10^4	9.47×10^{-3}	V/cm, $V/(K)^{c1}$
c_1	`hc1`	4.379	2.492	V/(cm K), 1
c_2	`hc2`	0.13	0	$V/(cm\ K^2)$, 1
d_0	`hd0`	1.234×10^6	1.404×10^6	V/cm
d_1	`hd1`	1.204×10^3	2.974×10^3	V/(cm K)
d_2	`hd2`	0.567	1.483	$V/(cm\ K^2)$

The model parameters are accessible in the parameter file section:

```
UniBo :{...}
```

9.9.6 Driving force

In DESSIS, the driving force F for impact ionization can be computed as the magnitude of the electrostatic field vector (keyword `ElectricField`), the component of the electrostatic field in the direction of the current (`Eparallel`), or the value of the gradient of the quasi-Fermi level (`GradQuasiFermi`) (see Table 15.10 on page 15.46). The default value is `GradQuasiFermi`.

The option `ElectricField` is used to perform breakdown simulations using the 'ionization integral' method (see Section 9.10 on page 15.220).

Table 15.89 Driving force models for avalanche breakdown

Transport model	Keyword	Driving force model

Drift-diffusion	`Electric Field`
	`Eparallel`
	`GradQuasiFermi` (default)
		$F = \lvert\vec{F}\rvert$
		$F = F_{\parallel}^{n,p} = \vec{F} \bullet (\overrightarrow{j_{n,p}} / \lvert j_{n,p}\rvert)$
		$F = \lvert\nabla\varphi_{n,p}\rvert$

Hydrodynamic	`CarrierTempDrive`	See Section 9.9.7.

9.9.7 Avalanche generation with hydrodynamic transport

If the hydrodynamic transport model is used, it is also possible (and usually recommended) to select a local carrier temperature–dependent impact ionization model. This is achieved by using the construct `Avalanche(CarrierTempDrive)` as an option to `Recombination(Avalanche)` in the `Physics` section of the DESSIS input file:

```
Physics{ Hydro
   Recombination(Avalanche(CarrierTempDrive)...}
```

Otherwise, the default electric field computation is still based on the `GradQuasiFermi` method.

The usual conversion of local carrier temperatures to effective fields E^{eff} is described by the algebraic equations:

$$n\mu_n\left(E_n^{\text{eff}}\right)^2 = n\frac{3k_B}{2q}\frac{T_n - T_L}{\lambda_n\tau_{en}} \tag{15.248}$$

$$p\mu_p\left(E_p^{\text{eff}}\right)^2 = p\frac{3k_B}{2q}\frac{T_p - T_L}{\lambda_p\tau_{ep}} \tag{15.249}$$

which are obtained from the energy conservation equation under time-independent, homogeneous conditions. (Eq. 15.248) and (Eq. 15.249) have been simplified in DESSIS by using the assumption $\mu_n E_n^{\text{eff}} = v_{sat,n}$ and $\mu_p E_p^{\text{eff}} = v_{sat,p}$, where $v_{sat,n}$ and $v_{sat,p}$ are the carrier saturation velocities. This assumption is true for high values of the electric field. However, for low field, the impact ionization rate is negligibly small.

The parameters λ_n and λ_p are fitting coefficients (default value 1) and their values can be changed in the parameter file, where they are represented as `n_l_f` and `p_l_f`, respectively, in the section:

```
AvalancheFactors { ... }
```

The conventional conversion formulas (Eq. 15.248) and (Eq. 15.249) can be activated by specifying parameters $\Upsilon_n = 0$, $\Upsilon_p = 0$ in the same `AvalancheFactors` section.

The simplified conversion formulas discussed above predict a linear dependence of effective electric field on temperature for high values of carrier temperature. For silicon, however, Monte Carlo simulations do not confirm this behavior. To obtain a better agreement with Monte Carlo data, additional heat sinks must be taken into account by the inclusion of an additional term in the equations for E^{eff}. Such heat sinks arise from nonelastic processes, such as the impact ionization itself. DESSIS supports the following model to account for these heat sinks:

$$nV_{sat,n}E_n^{\text{eff}} = n\frac{3k_B}{2q}\frac{T_n - T_L}{\lambda_n \tau_{en}} + \frac{\Upsilon_n}{q}(E_g + \delta_n k_B T_n)\alpha_n n v_{sat,n} \tag{15.250}$$

A similar equation E_p^{eff} is used to determine E_p^{eff}. To activate this model, set the parameters Υ_n and Υ_p to 1. This is the default for silicon, where the generalized conversion formula (Eq. 15.250) gives good agreement with Monte Carlo data for $\delta_n = \delta_p = 3/2$. For all other materials, the default of the parameters Υ_n and Υ_p is 0.

Table 15.90 Hydrodynamic avalanche model: Default parameters

Symbol	Parameter name	Default value
λ_n	`n_l_f`	1
λ_p	`p_l_f`	1
Υ_n	`n_gamma`	1
Υ_p	`p_gamma`	1
$\delta_n = 3/2$	`n_delta`	1.5
$\delta_n = 3/2$	`p_delta`	1.5

The effective fields E_n^{eff} and E_p^{eff} are then used as a driving force in the `Lackner`, `Okuto`, or `vanOverstraeten` formula (depending on the user option) for the impact ionization generation rate:

$$\alpha_n = \alpha_n(F) = \alpha_n\left(E_n^{\text{eff}}\right) \tag{15.251}$$

$$\alpha_p = \alpha_p(F) = \alpha_p\left(E_p^{\text{eff}}\right) \tag{15.252}$$

NOTE This procedure ensures that the same results are obtained as with the conventional local field–dependent models in the bulk case. Conversely, the temperature-dependent impact ionization model usually gives much more accurate predictions for the substrate current in short-channel MOS transistors.

9.10 Approximate breakdown analysis: Poisson equation approach

Junction breakdown due to avalanche generation is simulated by inspecting the ionization integrals:

$$I_n = \int_0^W \alpha_n(x) \; e^{-\int_x^W (\alpha_n(x') - \alpha_p(x')) \, dx'} \, dx \tag{15.253}$$

$$I_p = \int_0^W \alpha_p(x) \; e^{-\int_0^x (\alpha_p(x') - \alpha_n(x')) \, dx'} \, dx \tag{15.254}$$

where α_n , α_p are the ionization coefficients for electrons and holes, respectively, and W is the width of the depletion zone. The integrations are performed along field lines through the depletion zone. Avalanche breakdown occurs if an ionization integral equals one. (Eq. 15.253) describes electron injection (electron primary current) and (Eq. 15.254) describes hole injection. Since these breakdown criteria do not depend on current densities, a breakdown analysis can be performed by computing only the Poisson equation and ionization integrals under the assumption of constant quasi-Fermi levels in the depletion region.

9.10.1 Syntax and implementation

If only the Poisson equation is solved, it is important to use the parallel electric field (`Eparallel`) as the driving force in the avalanche generation rate because the parallel electric field and the gradients of the quasi-Fermi levels are not computed correctly:

```
Physics{ ...
   Avalanche(Eparallel)
   ComputeIonizationIntegrals ... }
```

`ComputeIonizationIntegrals` switches on the computation of the ionization integrals for ionization paths crossing the local field maxima in the semiconductor. By default, DESSIS reports only the path with the largest I_{mean}. With the addition of the keyword `WriteAll`, information about the ionization integrals for all computed paths is written to the log file.

The `Math` keyword `BreakAtIonIntegral` is used to terminate the quasistationary simulation when the largest ionization integral is greater than one.

The complete syntax of this keyword is `BreakAtIonIntegral(<number> <value>)` where a quasistationary simulation finishes if the `number` ionization integral is greater than `value`, and the ionization integrals are ordered with respect to decreasing value:

```
Math { BreakAtIonIntegral }
```

Three optional keywords in the `Plot` section specify the values of the corresponding ionization integrals that are stored along the breakdown paths:

```
Plot { eIonIntegral | hIonIntegral | MeanIonIntegral }
```

These ion integrals can be visualized by using Tecplot-ISE. A typical DESSIS command file is:

```
Electrode {
{ name="anode"   Voltage=0 }
   { name="cathode" Voltage=600 }
}
File {
   grid    = "@grid@"
   doping  = "@doping@"
   current = "@plot@"
   output  = "@log@"
   plot    = "@data@"
}
Physics {
   Mobility (DopingDep HighFieldSaturation)
   Recombination(SRH Auger Avalanche(Eparallel))
   ComputeIonizationIntegrals(WriteAll)
}
Solve {
   Quasistationary(
      InitialStep=0.02 MaxStep=0.01 MinStep=0.01
      Goal {name=cathode voltage=1000}
   )
   { poisson }
}
Math {
    Iterations=100
    BreakAtIonIntegral
}
Plot {
    eIonIntegral hIonIntegral MeanIonIntegral
    eDensity hDensity
    ElectricField/Vector
    eAlphaAvalanche hAlphaAvalanche
}
```

9.11 Band-to-band tunneling models

9.11.1 Schenk model

Phonon-assisted band-to-band tunneling cannot be neglected in steep pn-junctions (with a doping level of 1×10^{19} cm^{-3} or more on both sides) or in high normal electric fields of MOS structures. It must be switched on if the field, in some regions of the device, exceeds (approximately) 8×10^{5} V/cm. In this case, defect-assisted tunneling (see Section 2.5.2.3 on page 15.47 and Section 9.2 on page 15.204) must also be switched on.

Band-to-band tunneling is modeled using the expression [88]:

$$R_{\text{net}}^{\text{bb}} = AF^{7/2}\frac{\tilde{n}\tilde{p}-n_{\text{i, eff}}^{2}}{(\tilde{n}+n_{\text{i,eff}})(\tilde{p}+n_{\text{i,eff}})}\times\left[\frac{\left(F_c^{\mp}\right)^{-\frac{3}{2}}e^{-\frac{F_c^{\mp}}{F}}}{e^{\frac{\hbar\omega}{k_\text{B}T}}-1}+\frac{\left(F_c^{\pm}\right)^{-\frac{3}{2}}e^{-\frac{F_c^{\pm}}{F}}}{1-e^{-\frac{\hbar\omega}{k_\text{B}T}}}\right] \tag{15.255}$$

with the modified electron density:

$$\tilde{n} = n\left(\frac{n_{\mathrm{i,eff}}}{N_{\mathrm{C}}}\right)^{\frac{|\nabla E_{F_n}|}{F}} \tag{15.256}$$

and a similar relation for $\tilde{p}$, and with critical field strengths:

$$F_C^{\pm} = B(E_{\mathrm{g,eff}} \pm \hbar\omega)^{3/2} \tag{15.257}$$

The upper sign in (Eq. 15.255) refers to tunneling generation ($np < n_{\mathrm{i,eff}}^2$) and the lower sign refers to recombination ($np > n_{\mathrm{i,eff}}^2$). The quantity $\hbar\omega$ denotes the energy of the transverse acoustic phonon. For Fermi statistics and quantization, (Eq. 15.255) is modified in the same way as for SRH recombination (see (Eq. 15.201)).

Syntax and implementation

Band-to-band tunneling is switched on by using the keyword `Band2Band` in the `Recombination` statement in the `Physics` section (see Table 15.11 on page 15.47):

```
Physics { ...
   Recombination ( Band2Band )
}
```

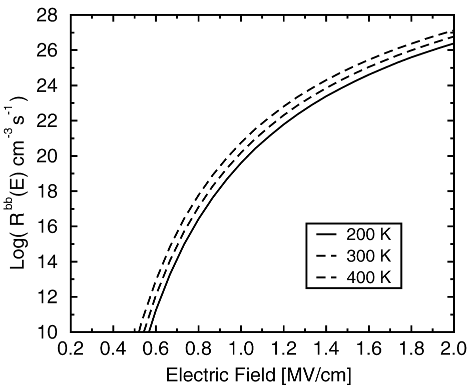


Figure 15.41 Band-to-band tunneling rate for different temperatures

Default parameters [88] are given in Table 15.91 and can be accessed in the parameter file section:

```
Band2BandTunneling { ...}
```

Table 15.91 Coefficients for band-to-band tunneling (Schenk model)

Symbol	Parameter name	Default value	Unit
A	`A`	8.977×10^{20}	$(\mathrm{cm\ s})^{-1}\ \mathrm{V}^{-2}$
B	`B`	2.14667×10^{7}	$(\mathrm{eV})^{-3/2}\ \mathrm{Vcm}^{-1}$
hω	`hbarOmega`	18.6	meV

These parameters were obtained assuming the field direction to be <111>, as in the case of defect-assisted tunneling.

9.11.2 Commonly used models

In addition to the Schenk band-to-band tunneling model, a set of simple and commonly used models is available. The difference between these commonly used models and the Schenk model is in the electric field dependence, which defines any physical effects in the tunneling (for example, direct or phonon-assisted tunneling). A general expression for the models can be written for generation term [114] as:

$$R^{\mathrm{b2b}} = AF^{\alpha}\exp\left(-\frac{B}{F}\right) \tag{15.258}$$

where F is the magnitude of the local electric field. The values of α that are available are $\alpha = 1$, $\alpha = 1.5$, and $\alpha = 2$.

Syntax and implementation

These band-to-band tunneling models are switched on by using the key parameters `E1`, `E1_5`, or `E2`, respectively. For example:

```
Physics { ...
   Recombination(Band2Band(E2)) }
```

switches on the simple model with $\alpha = 2$. Table 15.92 lists the coefficients of models and their defaults. The coefficients (A and B) can be changed in the parameter file in the section:

```
Band2BandTunneling { ... }
```

Table 15.92 Coefficients for band-to-band tunneling (commonly used models)

Keyword	α	A	B
`Band2Band(E1)`	$\alpha = 1$	$1.1\mathrm{e}27\,1/(\mathrm{cm}^2\cdot\mathrm{s}\cdot\mathrm{V})$	21.3e6 V/cm
`Band2Band(E1_5)`	$\alpha = 1.5$	$1.9\mathrm{e}24\,1/(\mathrm{cm}^{1.5}\cdot\mathrm{s}\cdot\mathrm{V}^{1.5})$	21.9e6 V/cm
`Band2Band(E2)`	$\alpha = 2$	$3.4\mathrm{e}21\,1/(\mathrm{cm}\cdot\mathrm{s}\cdot\mathrm{V}^2)$	22.6e6 V/cm

9.11.3 Hurkx model

The Hurkx band-to-band tunneling model [126] is implemented to provide a complete range of commonly used band-to-band tunneling models in DESSIS.

Similar to the other band-to-band tunneling models, the tunneling carriers are modeled by an additional generation–recombination contribution. In the Hurkx model, this contribution is expressed as:

$$G^{\mathrm{bb}} = -A\cdot D\cdot\left(\frac{F}{F_0}\right)^{P}\cdot\exp\left(-\frac{B\cdot E_{\mathrm{g}}(T)}{E_{\mathrm{g}}(300\mathrm{K})^{3/2}F}\right) \tag{15.259}$$

where F is the electric field, $F_0 = 1\mathrm{V/m}$, and:

$$D = \frac{np - n_{\mathrm{i,eff}}^2}{(n+n_{\mathrm{i,eff}})(p+n_{\mathrm{i,eff}})}(1-|\alpha|)+\alpha \tag{15.260}$$

Here, specifying α equal to 0 gives the original Hurkx model, whereas α equal to -1 gives only generation ($D = -1$), and α equal to $+1$ gives only recombination ($D = 1$). Therefore, if $D < 0$, it is a net carrier generation model. If $D > 0$, it is a recombination model. For Fermi statistics and quantization, (Eq. 15.260) is modified in the same way as for SRH recombination (see (Eq. 15.201)).

Syntax and implementation

The model is activated by the following keywords in the `Physics` section of the input file:

```
Physics { ...
   Recombination( Band2Band(Hurkx) )
}
```

Coefficients A [1/(cm^{3}s)], B [V/cm], and P are parameters of the model and can be specified in the following section of the DESSIS parameter file:

```
Band2BandTunneling {
   Agen = <value> [1/(cm3s)]
   Bgen = <value> [V/cm]
   Pgen = <value> [1]
   Arec = <value> [1/(cm3s)]
   Brec = <value> [V/cm]
   Prec = <value> [1]
   alpha = <value> [1]
}
```

In this parameter section, it is possible to specify different coefficients for the generation (`Agen`, `Bgen`, `Pgen`) and recombination (`Arec`, `Brec`, `Prec`) of carriers.

9.11.4 Tunneling near interfaces and equilibrium regions

Physically, band-to-band tunneling occurs over a certain tunneling distance. If the material properties or the electric field change significantly over this distance, (Eq. 15.255), (Eq. 15.258), and (Eq. 15.259) become inaccurate. In particular, near insulator interfaces, band-to-band tunneling vanishes, as no states to tunnel to are available in the insulator.

In some parts of the device (near equilibrium regions), it is possible that the electric field is large but changes rapidly, so that the actual tunneling distance (the distance over which the electrostatic potential change amounts to the band gap) is bigger and, therefore, tunneling is much smaller than expected from the local field alone.

To account for these effects, two additional control parameters are introduced in the `Band2BandTunneling` section of the DESSIS parameter file:

```
dDist = <value>      # [cm]
dPot = <value>       # [V]
```

By default, both these parameters equal zero. DESSIS disables band-to-band tunneling within a distance `dDist` from insulator interfaces. If `dPot` is nonzero (reasonable values for `dPot` are of the order of the band gap), DESSIS disables band-to-band tunneling at each point where the change of the electrostatic potential in field direction within a distance of `dPot`$/F$ (F is the local electric field) is smaller than `dPot`$/2$.

CHAPTER 10 Traps

DESSIS allows four types of traps (donor, acceptor, neutral electron, and neutral hole) and four types of DOS distributions (level, constant, exponential, and Gaussian). All trap models are available for both bulk semiconductors and interfaces. Trap distributions and characteristics are defined using either the `Traps` or `Amorphous` statements.

10.1 Trap energy and space distributions

The following expressions for trap concentration versus an energy (E) define different types of DOS:

$$\begin{array}{ll} N_0 \quad \text{at} \quad (E = E_0) & \text{Level} \\ N_0 \quad \text{for} \quad (E_0 - E_S < E < E_0 + E_S) & \text{Constant} \\ N_0 e^{-\left|\frac{E - E_0}{E_S}\right|} & \text{Exponential} \\ N_0 e^{-0.5\left(\frac{E - E_0}{E_S}\right)^2} & \text{Gaussian} \end{array} \qquad (15.261)$$

Figure 15.42 illustrates these distributions.

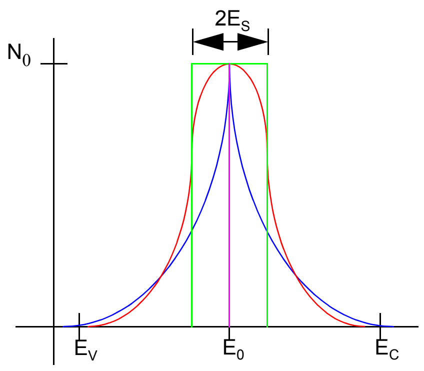


Figure 15.42 Trap energy distributions

For space-distributed traps, the user can specify any field stored in the doping file or PMI user fields (DF–ISE format), which DESSIS uses as input in the `File` section. For example, using MDRAW, the field `DeepLevels` can be written into the doping file. If `Sfactor = "DeepLevels"` is included in a `Traps` statement, DESSIS uses this field as the shape of the trap space distribution (the field is scaled by its maximum and multiplied by N_0).

10.2 Trap occupation dynamics

Trapped electron and hole concentrations on one energy level are related to occupation probabilities for electrons (f_n) and holes (f_p) as follows:

$$\begin{aligned} n_{\mathrm{Dt}} &= N_{\mathrm{Dt}} f_n \qquad & n_{\mathrm{t}} &= N_{\mathrm{Et}} f_n \\ p_{\mathrm{At}} &= N_{\mathrm{At}} f_p \qquad & p_{\mathrm{t}} &= N_{\mathrm{Ht}} f_p \end{aligned} \tag{15.262}$$

where N_{Dt} is the donor trap concentration, N_{At} is the acceptor trap concentration, N_{Et} is the neutral electron trap concentration, N_{Ht} is the neutral hole trap concentration, n_{Dt} is the electron concentration of the donor trap level, p_{At} is the hole concentration of the acceptor trap level, n_{t} is the electron concentration of the neutral electron trap level, and p_{t} is the hole concentration of the neutral hole trap level. Since $f_p = 1 - f_n$, only electron traps are considered to illustrate the general expressions for recombination processes.

In the case of energy-distributed traps, the band gap of the semiconductor is divided by an adaptive grid, and the trap concentration for each energy level E_{t} is calculated as an integral between these nodes. Each energy distribution defined in the input file has its own grid of energy levels. The default number of levels is 15, but the user can define this value in the `Math` statement `TrapDLN=<integer>` and can specify any number of energy levels and energy-distributed traps.

In the presence of traps, the Poisson equation is modified:

$$\nabla \cdot \varepsilon \nabla \phi = -q \cdot \left(p - n + N_{\mathrm{D}} - N_{\mathrm{A}} + \sum_{E_{\mathrm{t}}} (N_{\mathrm{Dt}} - n_{\mathrm{Dt}}) - \sum_{E_{\mathrm{t}}} (N_{\mathrm{At}} - p_{\mathrm{At}}) + \sum_{E_{\mathrm{t}}} p_{\mathrm{t}} - \sum_{E_{\mathrm{t}}} n_{\mathrm{t}} \right) \tag{15.263}$$

where $\sum_{E_{\mathrm{t}}} \cdot$ is a sum for all trap energy levels.

Users can plot total positive ($\sum_{E_{\mathrm{t}}} (N_{\mathrm{Dt}} - n_{\mathrm{Dt}}) + \sum_{E_{\mathrm{t}}} p_{\mathrm{t}}$) and negative ($\sum_{E_{\mathrm{t}}} (N_{\mathrm{At}} - p_{\mathrm{At}}) + \sum_{E_{\mathrm{t}}} n_{\mathrm{t}}$) trapped charges using the plotting keywords `eTrappedCharge` and `hTrappedCharge`, respectively.

10.2.1 Balance equation

The balance of the carrier flow to and from a trap level gives the following expression for the electron concentration of the trap level:

$$\frac{dn_{\mathrm{t}}}{dt} = v_{\mathrm{th}}^{n} \sigma_n N_{\mathrm{Et}} \left(\frac{n_1}{g_n} f_n - n(1 - f_n) \right) + v_{\mathrm{th}}^{p} \sigma_p N_{\mathrm{Et}} \left(\frac{p_1}{g_p} (1 - f_n) - p f_n \right) \tag{15.264}$$

where g_n and g_p are the electron and hole degeneracy factors, respectively, and:

$$n_1 = n_{\mathrm{i,eff}} \exp(E_{\mathrm{t}}/k_{\mathrm{B}}T), \quad p_1 = n_{\mathrm{i,eff}} \exp(-E_{\mathrm{t}}/k_{\mathrm{B}}T) \tag{15.265}$$

where E_{t} is a trap energy measured from the middle of the band gap, n and p are the free electron and hole concentrations, v_{th}^{n} and v_{th}^{p} are the electron and hole thermal velocities, σ_n and σ_p are the electron and hole capture cross sections, and T is the lattice temperature. The expressions for n_1 and p_1 above are valid for Boltzmann statistics and without quantization.

If Fermi–Dirac statistics or a quantization model (see Chapter 7 on page 15.165) is used, n_1 and p_1 are multiplied by the coefficients γ_n and γ_p defined in (Eq. 15.70) and (Eq. 15.71) (see Section 4.4 on page 15.137). In the balance equation (Eq. 15.264), generally, two terms are related to electron and hole flows. Correspondingly, each term has two components of carriers going to the trap level and returning to the band. For example, the term $n(1-f_n)$ reflects electron flow into the trap level and is proportional to a carrier concentration in the band and a concentration of empty trap levels.

Another term $\frac{n_1}{g_n}f_n$ gives the thermionic emission flow of electrons from the trap level to the conduction band.

These terms can be expressed differently for some special applications and models. For example, the radiation model (see Section 12.4 on page 15.242) modifies the carrier flow into the trap level.

10.2.2 Models for balance coefficients

All models of the balance coefficients of (Eq. 15.264) are mostly empirical or experimental, and are used for calibration purposes. The electron and hole thermal velocities have two formula options, which are provided through the DESSIS parameter file:

$$\texttt{VthFormula=1} \qquad v_{\mathrm{th}}^{n,p} = v_0^{n,p}\sqrt{\frac{T}{T_0}} \tag{15.266}$$

$$\texttt{VthFormula=1} \qquad v_{\mathrm{th}}^{n,p} = \left(3\frac{k\cdot T}{m_{300}^{n,p}}\right)^{0.5} \tag{15.267}$$

with T_0=300 K. The 300 K reference thermal velocities for the electrons and holes, v_0^n and v_0^p are 2.042×10^7 cm/s and 1.562×10^7 cm/s, respectively. m_{300}^n and m_{300}^p are the electron and hole DOS effective masses calculated by DESSIS at 300 K.

The electric field dependence of the cross sections $\sigma_{n,p}$ is selected by the `XsecFormula` parameter pair in the `Traps` parameter set in the DESSIS parameter file. The default value of `1` selects a field-independent cross section. A value of `2` selects the field-dependence for the J-model (see (Eq. 15.282)). A value of `3` selects the expression for the Hurkx trap-assisted tunneling model (see (Eq. 15.215)). A value of `4` selects a Poole–Frenkel model [153]:

$$\begin{aligned} \sigma_{n,p} &= \sigma_{n,p}^0(1+\Gamma_{\mathrm{pf}}) \\ \Gamma_{\mathrm{pf}} &= \frac{1}{\alpha^2}[1+(\alpha-1)\exp(\alpha)]-\frac{1}{2} \\ \alpha &= \frac{1}{kT}\sqrt{\frac{q^3E}{\pi\varepsilon_{\mathrm{pf}}}} \end{aligned} \tag{15.268}$$

The Poole–Frenkel model is frequently used for the interpretation of transport effects in dielectrics and amorphous films. The Poole–Frenkel theory predicts the enhanced emission probability Γ_{pf} for Coulomb trap centers where the potential barrier is reduced because of the high external electric field. Remembering that it occurs only at Coulomb centers, this model works differently for donor and acceptor centers.

For `Donor` and `hNeutral` traps, the electron and hole cross sections are computed as follows: $\sigma_n = \sigma_n^0(1+\Gamma_{\mathrm{pf}})$ and $\sigma_p = \sigma_p^0$. Likewise for `Acceptor` and `eNeutral` types, the enhanced emission probability applies only to the hole cross section: $\sigma_p = \sigma_p^0(1+\Gamma_{\mathrm{pf}})$ and $\sigma_n = \sigma_n^0$.

The Poole–Frenkel model has only one parameter $\varepsilon_{\mathrm{pf}}$, which can be specified in the following section of the DESSIS parameter file:

```
PooleFrenkel
{ * TrapXsection = Xsec0*(1+Gpf)
  * Gpf = (1+(a-1)*exp(a))/a^2-0.5
  * where
  *       a = (1/kT)*(q^3*F/pi/e0/epsPF)^0.5,
  *       F is the electric field.
    epsPF = 11.7 ,11.7 # [1]
}
```

All other coefficients of the above models can be specified in the `Traps` section of the DESSIS parameter file as follows:

```
Traps:
{
    *   G is degeneracy factor
        G = 1 , 1                                       # [1]
        Xsec = 1.0000e-15 , 1.0000e-15                  # [cm^2]
    *   VthFormula=1: Vth(T) = Vth*(T/300)^1/2
    *   VthFormula=2: Vth(T) = (3*k*T/m_300)^1/2
        VthFormula = 1 , 1                              # [1]
        Vth = 2.0420e+07 , 1.5626e+07                   # [cm/s]
    *   XsecFormula=1: Xsec(F) = Xsec
    *   XsecFormula=2: Xsec(F) = Xsec*(1+a1*(F/F0)^p1+a2*(F/F0)^p2)^p0
    *   XsecFormula=3: Xsec(F) = Xsec*(1+Gt), Gt is Hurkx TATunneling factor
    *   XsecFormula=4: Xsec(F) = Xsec*(1+Gpf), Gpf is Poole-Frenkel factor
        XsecFormula = 1, 1
```

Values from the parameter file are used as defaults for trap definition in the `Physics` section of the input file. For example, if the user does not specify cross sections in the input file, the values are taken from the parameter file.

`VthFormula` and `XsecFormula` can be different for electrons and holes. It is also possible to specify these models for each trap distribution separately in the input file and to have any combination of the electric field dependence models, see Table 15.93 on page 15.230.

10.3 Steady state analysis

In the steady state, $dn_{\mathrm{t}}/dt = 0$. Therefore, for one level and $g_n = g_p = 1$ (to simplify expressions):

$$f_n = \frac{v_{\mathrm{th}}^n \sigma_n n + v_{\mathrm{th}}^p \sigma_p p_1}{v_{\mathrm{th}}^n \sigma_n (n + n_1) + v_{\mathrm{th}}^p \sigma_p (p + p_1)} \tag{15.269}$$

The recombination process through one level is the same for electrons and holes, and the recombination term can be written as:

$$R_{\mathrm{Et}} = N_{\mathrm{Et}} \frac{v_{\mathrm{th}}^n \sigma_n v_{\mathrm{th}}^p \sigma_p (np - n_{\mathrm{i,eff}}^2)}{v_{\mathrm{th}}^n \sigma_n (n + n_1) + v_{\mathrm{th}}^p \sigma_p (p + p_1)} \tag{15.270}$$

If the lifetimes for electrons and holes are defined as $\tau_p = 1/v_{\text{th}}^p \sigma_p N_{\text{Et}}$ and $\tau_n = 1/v_{\text{th}}^n \sigma_n N_{\text{Et}}$ respectively, the recombination term can be rewritten in standard SRH form:

$$R_{\text{Et}} = (np - n_{\text{i,eff}}^2)/(\tau_p(n+n_1) + \tau_n(p+p_1)) \tag{15.271}$$

Each level adds a term. Therefore, in the presence of traps, the trap total recombination term in the continuity equations is the sum for all levels:

$$R_{\text{Trap}} = \sum_{E_{\text{t}}} R_{\text{Dt}} + \sum_{E_{\text{t}}} R_{\text{At}} + \sum_{E_{\text{t}}} R_{\text{Ht}} + \sum_{E_{\text{t}}} R_{\text{Et}} \tag{15.272}$$

Users can plot R_{Trap} using plotting keywords for electrons (`eGapStatesRecombination`) or holes (`hGapStatesRecombination`), which will give the same data in the steady-state simulation cases.

10.4 Transient analysis

In the transient case, the differential equations for the electron-related occupation probability at each energy level are:

$$\frac{df_n}{dt} + \left[v_{\text{th}}^n \sigma_n \left(n + \frac{n_1}{g_n}\right) + v_{\text{th}}^p \sigma_p \left(p + \frac{p_1}{g_p}\right)\right] f_n = v_{\text{th}}^n \sigma_n n + v_{\text{th}}^p \sigma_p \frac{p_1}{g_p} \tag{15.273}$$

These equations are solved self-consistently with the transport and Poisson equations. In the transient case, the recombination processes for electrons and holes are different, and the SRH form cannot be applied.

For one level, this becomes:

$$\begin{aligned} R_{\text{Et}}^n &= v_{\text{th}}^n \sigma_n N_{\text{Et}} \left[n(1-f_n) - \frac{n_1}{g_n} f_n\right] \\ R_{\text{Et}}^p &= v_{\text{th}}^p \sigma_p N_{\text{Et}} \left[p f_n - \frac{p_1}{g_p}(1-f_n)\right] \end{aligned} \tag{15.274}$$

The total recombination terms for the electron and hole continuity equations are also equal to the sum for all levels as in the steady state case. Users can plot the total recombination rates for electrons and holes using the plotting keywords `eGapStatesRecombination` and `hGapStatesRecombination`, respectively.

10.5 Syntax for traps

To specify bulk traps, the `Traps` statement must be defined in the `Physics` section of the DESSIS input file. The complete list of keywords and parameters of the `Traps` section is given in Table 15.93 on page 15.230. The most commonly used ones are:

```
Physics{ Traps(
   Donor | Acceptor | eNeutral | hNeutral
   Level | Uniform | Exponential | Gaussian
   fromCondBand | fromValBand | fromMidBandGap
   Conc = No
   EnergyMid = Eo
   EnergySig = Es
   eXsection = σn
```

```
      hXsection = σ_p
      eGfactor = g_n
      hGfactor = g_p
      )
   }
```

Interface traps are defined in the same way in the `MaterialInterface` or `RegionInterface Physics` section:

```
Physics( MaterialInterface="<material1>/<material2>")
   { Traps( ... ) ... }
```

or:

```
Physics( RegionInterface="<region1>/<region2>")
   { Traps( ... ) ... }
```

NOTE For interface traps, the unit of the parameter `Conc` is cm^{-2}.

If many trap levels are required, each level must be defined separately inside the `Traps` statement in parentheses, for example, `Traps ((...) (...))`.

The following example of a trap statement illustrates one donor trap level in the middle of the band gap with a concentration of $1\times10^{15}\text{cm}^{-3}$ and capture cross sections of $1\times10^{-14}\text{cm}^{2}$:

```
Traps(Donor Level EnergyMid=0 fromMidBandGap
   Conc=1e15 eXsection=1e-14 hXsection=1e-14)
```

This example illustrates the DOS definition in a polysilicon TFT with four exponential distributions:

```
Traps( (eNeutral Exponential fromCondBand Conc=1e21 EnergyMid=0
   EnergySig=0.035 eXsection=1e-10 hXsection=1e-12)
   (eNeutral Exponential fromCondBand Conc=5e18 EnergyMid=0
   EnergySig=0.1 eXsection=1e-10 hXsection=1e-12)
   (hNeutral Exponential fromValBand Conc=1e21 EnergyMid=0
   EnergySig=0.035 eXsection=1e-12 hXsection=1e-10)
   (hNeutral Exponential fromValBand Conc=5e18 EnergyMid=0
   EnergySig=0.2 eXsection=1e-12 hXsection=1e-10) )
```

Table 15.93 Keyword options for Traps command

Keyword	Description

`Donor`
`Acceptor`
`eNeutral`
`hNeutral`
`FixedCharge`
	Defines the donor type of the traps.
	Defines the acceptor type of the traps.
	Defines the neutral electron type of the traps.
	Defines the neutral hole type of the traps.
	Defines the fixed charge.

`Level`
`Uniform`
`Exponential`
`Gaussian`
	Defines a one-energy trap level.
	Defines a uniform DOS.
	Defines an exponential DOS.
	Defines a Gaussian DOS.

`fromCondBand`
`fromValBand`
`fromMidBandGap`
	Defines zero energy (`CondBand`) and `ValBand` energy direction.
	Defines zero energy (`ValBand`) and `CondBand` energy direction.
	Defines zero energy (`MidBandGap`) and `CondBand` energy direction.

`Conc = <float>`	Defines the energy level concentration (cm^{-3} for bulk and cm^{-2} for interfaces), or the peak of the DOS ([cm^{-3} eV^{-1}] for bulk and [cm^{-2} eV^{-1}] for interfaces).

Table 15.93 Keyword options for Traps command

Keyword	Description
`EnergyMid = <float>`	Defines the DOS medium energy or level energy E_0 [eV] . At an interface between two semiconductors with different band gap energies, interface trap energy levels are referenced to the smaller band gap.
`EnergySig = <float>`	Defines the DOS energy sigma E_S [eV].

`eXsection = <float>`
`hXsection = <float>`
	Defines the electron capture cross section σ_n [cm^2].
	Defines the hole capture cross section σ_p [cm^2].

`ElectricField`
`Tunneling(Hurkx)`
 	
`PooleFrenkel`
	Activates XsecFormula = 2 for the capture cross section.
	Activates XsecFormula = 3 for the capture cross section (Hurkx trap-assisted tunneling model).
	Activates XsecFormula = 4 for the capture cross section (Poole–Frenkel model).

`eGfactor = <float>`
`hGfactor = <float>`
	Defines the electron degeneracy factor g_n .
	Defines the hole degeneracy factor g_p .

`eJfactor = <float>`
`hJfactor = <float>`
	Defines the electron J-model factor (radiation model).
	Defines the hole J-model factor (radiation model).

`Sfactor = <string>`	Defines a shape of trap space distribution by any internal DESSIS field. For example, it can be `DeepLevels`, `PMIUserField`, `xMoleFraction`.
`Sfactor = <PMI model name>`	The shape of the trap space distribution is computed by a PMI model. The name of this model must be different from the name of all internal DESSIS fields. The PMI is described in Section 33.25 on page 15.591.
`Add2TotalDoping`	Adds the energy level concentration to the total doping, donor, and acceptor concentrations used for a computation of the mobility, lifetimes, and so on (see Section 2.14 on page 15.95 where the total doping is defined).

10.6 Syntax for amorphous statement

For polysilicon devices, four exponential distributions are often sufficient to describe the DOS. In this case, traps can be defined equivalently by using the simplified `Amorphous` statement. Table 15.94 lists all of the parameters of the amorphous option. For the previous example of a DOS definition in a polysilicon TFT, the equivalent `Amorphous` statement can be:

```
Amorphous(DonPeakTail = 1e21 DonEnergyTail = 0.035
    DonPeakDeep = 5e18 DonEnergyDeep = 0.1
    eDonSigma = 1e-10 hDonSigma = 1e-12
    AccPeakTail = 1e21 AccEnergyTail = 0.035
    AccPeakDeep = 5e18 AccEnergyDeep = 0.2
    eAccSigma = 1e-12 hAccSigma = 1e-10)
```

Table 15.94 Keyword options for Amorphous command

Keyword	Description
`AccPeakTail = <float>`	Defines the peak of the exponential for acceptor tail states DOS [cm^{-3} eV^{-1}].
`AccEnergyTail = <float>`	Defines the characteristic energy of the exponential for acceptor tail states DOS [eV].
`AccPeakDeep = <float>`	Defines the peak of the exponential for acceptor deep states DOS [cm^{-3} eV^{-1}].
`AccEnergyDeep = <float>`	Defines the characteristic energy of the exponential for acceptor deep states DOS [eV].

Table 15.94 Keyword options for Amorphous command

Keyword	Description
`DonPeakTail = <float>`	Defines the peak of the exponential for donor tail states DOS [cm^{-3} eV^{-1}].
`DonEnergyTail = <float>`	Defines the characteristic energy of the exponential for donor tail states DOS [eV].
`DonEnergyDeep = <float>`	Defines the characteristic energy of the exponential for donor deep states DOS [eV].
`eAccSigma = <float>`	Defines the electron capture cross section for acceptor states [cm^2].
`hAccSigma = <float>`	Defines the hole capture cross section for acceptor states [cm^2].
`eDonSigma = <float>`	Defines the electron capture cross section for donor states [cm^2].
`hDonSigma = <float>`	Defines the hole capture cross section for donor states [cm^2].

10.7 Setting and unsetting an initial trap occupation

For the investigation of, for example, time-delay effects, it may be advantageous to start a transient simulation from an initial state with either totally empty or totally filled trap states. However, depending on the position of the equilibrium Fermi level it may be impossible to reach such an initial state from steady state/ quasistationary simulations (for example, it may be a metastable state with a very long lifetime). For this reason, new statements `Set` and `Unset` for the trap filling have been introduced into the `Solve` section. This statement is specified before any appropriate `Solve` statement, such as `Transient`, `Quasistationary`, or `Coupled`. The syntax of the statements is given in the Table 15.95.

Table 15.95 Keyword options for Set and Unset of trap occupation

Keyword	Description
`Set(TrapFilling = n)`	Defines a steady state trap occupation, which corresponds to the case when the Fermi level is *much higher* than the conduction band level (for example, it corresponds to the case of high electron concentration).
`Set(TrapFilling = p)`	Defines a steady state trap occupation, which corresponds to the case when the Fermi level is *much lower* than the valence band level (for example, it corresponds to the case of high hole concentration).
`Set(TrapFilling = Full)`	Provides maximum trap charge. (For `eNeutral` and `hNeutral` traps, it means that the trapped carrier concentration is equal to the total trap concentration, but for `Donor` and `Acceptor`, it means no carriers are trapped. As a result, it provides the maximum trap charge.)
`Set(TrapFilling = Empty)`	Provides zero trap charge. (For `eNeutral` and `hNeutral` traps, it means no carriers are trapped, but for `Donor` and `Acceptor`, it gives maximum possible carrier concentration. As a result, it provides the zero trap charge.)
`Set(TrapFilling = 0)`	Provides an occupation that corresponds to zero electron and hole concentrations.
`Set(TrapFilling = Frozen)`	Provides an unchanged trap occupation for a device characterization, for example, to compute the threshold voltage.
`Set(TrapFilling = -Degradation)`	Removes a modification in trap concentration due to the degradation model, and returns it to initial values (see Section 11.4 on page 15.238).
`UnSet(TrapFilling)`	Uses the standard trap equations.

Therefore, the `Solve` statement for such requirements can be:

```
Solve{
   Set(TrapFilling=n)
   Quasistationary{...}
   Unset(TrapFilling)
   Transient{...}
}
```

10.8 Numeric parameters

When used with Fermi statistics, the trap model sometimes leads to convergence problems, especially at the beginning of a simulation when DESSIS tries to find an initial solution. This problem can often be solved by changing the numeric damping of the trap charge in the nonlinear Poisson equation. To this end, set the `Damping` option to the `Traps` keyword in the global `Math` section to a nonnegative number, for example:

```
Math {
   Traps(Damping=100)
}
```

Larger values of `Damping` increase damping of the trap charge; a value of `0` disables damping. The default value is `10`.

Depending on the particular example, increasing damping can improve or degrade the convergence behavior. Currently, ISE has no guidelines regarding the optimal value of this parameter.

CHAPTER 11 Degradation model

11.1 Overview

A necessary part of CMOS reliability prediction is the simulation of the time dependence of interface trap generation. To cover as wide a range as possible, this simulation should accurately reflect the physics of the interface trap formation process. Disorder-induced variations among the Si-H activation energies at the passivated Si-SiO_2 interface have been shown [166] to be a plausible source of the sublinear time dependence of this trap generation process. Previously, diffusion of hydrogen from the passivated interface was used to explain some time dependencies [167]. However, it is considered here that this could be due to a Si-H density–dependent activation energy, which may be due to the effects of Si-H breaking on electrical and chemical potential of hydrogens at the interface. In addition, the field dependence of the activation energy (Poole–Frenkel effect) is included, so that all these factors lead to an enhanced trap formation kinetics.

11.2 Trap formation kinetics

The main assumption about trap formation is that initially dangling Si bonds at the Si–SiO_2 interface were passivated by H or D [168], and degradation is a depassivation process where hot carrier interactions with Si-H/D bonds or other mechanisms are responsible for this. The equations of the model are solved self-consistently with all transport equations.

11.2.1 Power law and kinetic equation

The experimental data for the kinetics of interface trap formation [170] shows that the time dependence of trap generation can be described by a simple power law: $N_{it} - N_{it}^0 = n_0/(1 + (kt)^{-\alpha})$, where N_{it} is the concentration of interface traps, and n_0 and N_{it}^0 are the initial concentrations of Si-H bonds and interface traps, respectively. Assuming $N = n_0 + N_{it}{}^0$ total Si bonds at the interface, the remaining number of Si-H bonds at the interface after stress is $n = N - N_{it}$ and follows the power law:

$$n = \frac{n_0}{1 + (kt)^{\alpha}} \qquad (15.275)$$

Based on experimental observations, the power α is stress dependent and varies between 0 and 1.

From first-order kinetics [166], it is expected that the Si-H concentration during stress obeys:

$$\frac{dn}{dt} = -k \cdot n \qquad (15.276)$$

where k is a reaction constant that can be described by $k = k_0 \exp(-\varepsilon_A / k_B T)$ in the Arrhenius approximation, ε_A is the Si-H activation energy, and T is the Si-H temperature. The exponential kinetics given by this equation ($n = n_0 \exp(-kt)$) do not fully describe the experimental data because a constant activation energy will behave like the power law in the first equation, but with power $\alpha \approx 1$.

11.2.2 Si-H density–dependent activation energy

This section describes an activation energy parameterization to capture the sublinear power law for the time dependence of interface trap generation. There is evidence that the hydrogen atoms, when removed from the silicon, remain negatively charged [169]. If this correct, the hydrogen can be expected to remain in the vicinity of the interface and will affect the breaking of additional silicon-hydrogen bonds by changing the electrical potential.

The concentration of released hydrogen is equal to $N-n$, so the activation energy dependence (assuming the activation energy changes exponentially with the breaking of Si-H) can be expressed as:

$$\varepsilon_A = \varepsilon_A^0 + \beta \cdot k_B T \ln\left(\frac{N-n}{N-n_0}\right) \tag{15.277}$$

where the last term represents the Si-H density–dependent change with a prefactor β. Note that $(N-n)/(N-n_0)$ is the fraction of traps generated to the total initial traps, and this gives the form of the chemical potential of Si-H bonds with prefactor β.

The numeric solution of the kinetic equation with the varying activation energy above clearly shows that such a Si-H density–dependent activation energy gives a power law, and the power α is a function of the prefactor β. From the available experimental data of interface trap generation, it was noted that for negative gate biases $\alpha > 0.5$, but for positive ones $\alpha < 0.5$. It is interesting that the solution of the above kinetic equation gives $\alpha \approx 0.5$ in the equilibrium case where a unity prefactor is used. In nonequilibrium, a polarity-dependent modification of the prefactor (field stretched and pressed Si-H bonds) is possible.

11.3 Syntax and parameterized equations

The degradation model based on the mentioned equations can be activated for any trap level or distribution (see Section 10.5 on page 15.229), and its keywords described in Table 15.96 on page 15.237 can be used with any other trap options listed in Table 15.93 on page 15.230.

The parameterized system of equations for the reaction constant k based on the trap formation model (see Section 11.2 on page 15.235) and [171] can be expressed as:

$$\begin{aligned}
k &= k_0 \exp\left(\frac{\Delta\varepsilon_A}{\varepsilon_T}\right)\exp\left(\frac{\varepsilon_A}{k_B T_0} - \frac{\varepsilon_A}{\varepsilon_T}\right) k_{FN} k_{HC} \\
\varepsilon_T &= k_B T + \delta_{//}|F_{//}|^{\rho_{//}} \\
k_{HC} &= 1 + \delta_{HC}|I_{HC}|^{\rho_{HC}} \\
k_{FN} &= 1 + \delta_{Tun}|I_{Tun}|^{\rho_{Tun}} \\
\Delta\varepsilon_A &= \delta_{\perp}|F_{\perp}|^{\rho_{\perp}} + (1+\beta)\varepsilon_T \ln\frac{N-n}{N-n_0} \\
\beta &= \beta_0 + \beta_{\perp}F_{\perp} + \beta_{//}F_{//}
\end{aligned}
\qquad
\begin{aligned}
k &= k_0 \exp\left(\frac{\Delta\varepsilon_A}{k_B T}\right)\exp\left(\frac{\varepsilon_A}{k_B T_0} - \frac{\varepsilon_A}{k_B T}\right) \\
\Delta\varepsilon_A &= \delta|F|^{\rho}
\end{aligned}
\tag{15.278}$$

where the left column of expressions is related to interface traps and the right (simplified), to bulk traps. k_0 is the reaction (depassivation) constant at the passivation equilibrium (for the passivation temperature T_0 and for no changes in the activation energy $\Delta\varepsilon_A = 0$). $F_{\perp}, F_{//}$ are perpendicular and parallel components of the electric field F to the interface where traps are located. I_{HC}, I_{Tun} are the local densities of hot carrier and tunneling (Fowler–Nordheim and direct) currents at the interface where traps are located. These currents represent hot carrier (see Chapter 17 on page 15.317) and tunneling stresses (see Chapter 16 on page 15.299) and should be activated in the `GateCurrent` model.

ε_T is the energy of hydrogen on Si-H bonds and is equal to $k_B T$ plus some possible gain from hot carriers represented as the additional term that is dependent on the parallel component of the electric field $\delta_{//}|F_{//}|^{\rho_{//}}$. $\Delta\varepsilon_A$ is a decrease of the activation energy because of stretched Si-H bonds [172] by the electric field (first term) and due to a change of the chemical potential (second term) [171]. Effectively, the influence of the chemical potential also can be different in the presence of the electric field, and the coefficient β represents this. Coefficients $\delta_\perp, \rho_\perp, \delta_{//}, \rho_{//}$, $\delta_{HC}, \rho_{HC}, \delta_{Tun}, \rho_{Tun}$, and $\beta_0, \beta_\perp, \beta_{//}$ are field and current enhancement parameters of the model. Table 15.96 lists the options available for the degradation model.

Table 15.96 Keyword options for degradation model

Keyword	Description

`Degradation`
`Degradation(PowerLaw)`
	The first keyword switches on the model based on the kinetic equation and the second activates the model based on the power law.

`DePasCoef` = k_0 [1/s]	Defines the depassivation coefficient at the passivation conditions (the equilibrium at the passivation temperature T_0).
`PasCoef` = γ_0 [1/s]	By default (no specification), this passivation coefficient is computed automatically to provide the equilibrium: ($\gamma_0 = \frac{n_0}{N-n_0}k_0$). The user can specify it directly.
`PasTemp` = T_0 [K] `PasVolume` = ν [cm^2] or [cm^3]	Defines the passivation temperature (at the equilibrium that appears in passivation process, by default T_0=300) and the passivation volume (see below, by default ν=0).
`ActEnergy` = ε_A [eV]	Defines the activation energy of hydrogen on Si-H bonds.
`BondConc` = N [cm^{-2}] or [cm^{-3}] `CritConc` = N_{crit} [cm^{-2}] or [cm^{-3}]	Defines the total Si dangling bond concentration (maximum trap concentration that can be reached due to degradation) and the critical trap concentration that will be used for device lifetime estimations (by default $N_{crit}=N/10$). For trap levels, this is a straightforward definition, but for energy distributed traps $N(\varepsilon)$, it controls the following integral $\int_{E_V}^{E_C} N(\varepsilon)d\varepsilon$.

`FieldEnhan`($\delta_{//}\rho_{//}\delta_\perp\rho_\perp$)
$\left[eV\left(\frac{V}{cm}\right)^{-\rho_{//}}\right]$ [1]
$\left[eV\left(\frac{V}{cm}\right)^{-\rho_\perp}\right]$ [1]
	Controls the parameters of the electric field–dependent terms of the Si-H bond energy and the activation energy, by default:
	`FieldEnhan(0 1 0 1)`.

`CurrentEnhan` ($\delta_{Tun}\rho_{Tun}\delta_{HC}\rho_{HC}$) $\left[\left(\frac{A}{cm^2}\right)^{-\rho_{Tun}}\right]$ [1] $\left[\left(\frac{A}{cm^2}\right)^{-\rho_{HC}}\right]$ [1]	Controls the parameters of the tunneling and hot carrier–dependent terms of the depassivation constant, by default: `CurrentEnhan(0 1 0 1)`.

Table 15.96 Keyword options for degradation model

Keyword	Description
PowerEnhan($\beta_0 \beta_\perp \beta_{//}$) [1] $\left[\left(\frac{V}{cm}\right)^{-1}\right] \left[\left(\frac{V}{cm}\right)^{-1}\right]$	Controls the parameters in the chemical potential for kinetic equation degradation and the power for degradation by power law, by default: PowerEnhan(0 0 0).

The following general kinetic equations (LHS, activated by the keyword `Degradation`) with an added passivation term and power law (right, activated by the keyword `Degradation(PowerLaw)`) are:

$$\begin{aligned} \frac{dn}{dt} &= -k \cdot n + \gamma(N-n) \\ \gamma_0 &= \frac{n_0}{N-n_0} k_0 \\ \gamma &= \gamma_0[1 + \nu(n_0 - n)] \end{aligned} \qquad \begin{aligned} n &= \frac{n_0}{1+(kt)^\alpha} \\ \alpha &= 0.5 + \beta \end{aligned} \tag{15.279}$$

where γ_0 is the passivation constant, which is computed automatically by default, to provide the equilibrium, but the user can specify it directly in the input file. ν is the passivation volume (by default, it is equal to zero), and it reflects the effect that depassivated hydrogen can be trapped by Si dangling bonds again. These depassivated hydrogens increase the average hydrogen concentration near traps.

11.4 Device lifetime and simulation

Based on Section 11.3 on page 15.236, the `Physics` section of a DESSIS input file that describes the parameters of the degradation model can be:

```
Physics(MaterialInterface="Silicon/Oxide"){
   Traps(Conc=1e8 EnergyMid=0 Acceptor #FixedCharge
      Degradation #(PowerLaw)
         ActEnergy=2 BondConc=1e12
      DePasCoeff=8e-10
      FieldEnhan=(0 1 1.95e-3 0.33)
      CurrentEnhan=(0 1 6e+5 1)
      PowerEnhan=(0 0 -1e-7)
   )
   GateCurrent(GateName="gate" Lucky(CarrierTemperatureDrive) Fowler)
}
```

For this input, the initially specified trap concentration is 10^8 cm^{-2} and, in the process of degradation, it can be increased up to 10^{12} cm^{-2}. The activation energy of hydrogen on Si-H bonds is 2 eV and the depassivation constant at the equilibrium is equal to 8 x 10^{-10} s^{-1}. The degradation simulation can be separated into two parts:

- Simulation of extremely stressed devices with existing experimental data and fitting to the data by modification of the field and current-dependent parameters.
- Simulation of normal-operating devices to predict device reliability (lifetime).

For the first part of the degradation simulation, the typical `Solve` section can be:

```
Solve {
   NewCurrentPrefix="tmp"
   coupled (iterations=100) { Poisson }
   coupled { poisson electron hole }
   Quasistationary( InitialStep=0.1 MaxStep=0.2 MinStep=0.0001
                  increment=1.5 Goal{name="gate" voltage=-10} )
                  { coupled { poisson electron hole } }
    NewCurrentPrefix=""
    coupled { poisson electron hole }
    transient( InitialTime=0 Finaltime = 100000
                  increment=2 InitialStep=0.1 MaxStep=100000 ){
               coupled{ poisson electron hole }
    }
}
```

The first `Quasistationary` ramps the device to stress conditions (in this particular case, to high negative gate voltage), the second `transient` simulates the degradation kinetics (up to 10^5 s, which is a typical time for stress experimental data).

NOTE The hot carrier currents are postprocessed values and, therefore, `InitialStep` should not be large.

To monitor the trap formation kinetics in transient, the user can use `Plot` and `CurrentPlot` statements to output `TotalInterfaceTrapConcentration`, `TotalTrapConcentration`, and `OneOverDegradationTime`, for example:

```
CurrentPlot{
   eDensity(359) Potential(359)
   eInterfaceTrappedCharge(359) hInterfaceTrappedCharge(359)
   OneOverDegradationTime(359) TotalInterfaceTrapConcentration(359)
}
```

where a vertex number is specified to have a plot of the fields at some location on the interface. As a result, the behavior of these values versus time can be seen in the DESSIS plot file.

The prediction of the device lifetime can be performed in two different ways:

- Direct simulation of a normal-operating device in transient for a long time (for example, 30 years).
- Extrapolation of the degradation of a stressed device by computation of the ratio between depassivation constants for stressed and unstressed conditions.

The important value here is the critical trap concentration N_{crit}, which defines an edge between a properly working and improperly working device. Using N_{crit}, the device lifetime τ_D is defined as follows (according to the two different prediction ways):

1. In `transient`, direct computation of time t=τ_D gives the trap concentration equal to N_{crit}.
2. In `Quasistationary`, if the previously finished `transient` statement computes the device lifetime τ_D^{stress} and the depassivation constant k^{stress} at stress conditions, then $\tau_D = (k^{stress}/k)\tau_D^{stress}$.

So, the plotted value of `OneOverDegradationTime` is equal to $1/\tau_D$ for one trap level and the sum $\sum 1/\tau_D^i$ if several trap levels are defined for the degradation. It is computed for each vertex where the degradation model is applied and can be considered as the lifetime of local device area.

For the second approach to device lifetime computation, the following `solve` statement can be used:

```
Solve {
   NewCurrentPrefix="tmp"
   coupled (iterations=100) { Poisson }
   coupled { poisson electron hole }

   Quasistationary( InitialStep=0.1 MaxStep=0.2 Minstep=0.0001
                increment=1.5 Goal{name="gate" voltage=-10} )
                { coupled { poisson electron hole } }

   NewCurrentPrefix=""
   coupled { poisson electron hole }
   transient( InitialTime=0 Finaltime = 100000
              increment=2 InitialStep=0.1 MaxStep=100000 ){
              coupled{ poisson electron hole }
   }

   set(Trapfilling=-Degradation)

   coupled { poisson electron hole }
   Quasistationary( InitialStep=0.1 MaxStep=0.2 Minstep=0.0001 increment=1.5
                Goal{name="gate" voltage=1.5} )
                { coupled { poisson electron hole } }
   Quasistationary( InitialStep=0.1 MaxStep=0.2 Minstep=0.0001 increment=1.5
                Goal{name="drain" voltage=3} )
                { coupled { poisson electron hole } }
}
```

The statement `set(Trapfilling=-Degradation)` returns the trap concentrations to their unstressed values (it is not necessary to include, but it may be interesting to check the influence). The first `Quasistationary` statement after the `set` command returns the normal-operating voltage on the gate and, in the second, the user can plot the dependence of $1/\tau_D$ on applied drain voltage. The last dependence could be useful to predict an upper limit of operating voltages where the device will work for a specified time.

CHAPTER 12 Radiation models

12.1 Overview

DESSIS allows for the simulation of degradation of semiconductor devices due to received radiation. For now, this degradation is modeled as a change of trapped charge, which may cause a shift in device characteristics. Usually, degradation is important in insulators (for example, oxide) and users should define these insulators as wide band gap semiconductors so that the appropriate transient trap equations can be solved inside these regions.

12.2 Syntax and implementation

The radiation model is activated by specifying the keyword `Radiation(...)` (with optional parameters) in the `Physics` section of the input file:

```
Radiation{
   Dose = <value> | DoseRate = <value>
   DoseTime = (<value>,<value>)
   DoseSigma = <value>
}
```

where `DoseRate` [rad/s] represents D in (Eq. 15.280). The optional `DoseTime` [s] allows the user to specify the time period during which exposure to the constant `DoseRate` occurs. `DoseTSigma` [s] can be combined with `DoseTime` to specify the standard deviation of a Gaussian rise and fall of the radiation exposure.

As an alternative to `DoseRate`, `Dose` [rad] can be specified to represent the total radiation exposure over the `DoseTime` interval. In this case, `DoseTime` must be specified.

12.3 Yield function

Generation of electron–hole pairs due to radiation is an electric field–dependent process [123] and is modeled as follows:

$$
\begin{aligned}
G_r &= g_0 D \cdot Y(E) \\
Y(E) &= \left(\frac{|E| + E_0}{|E| + E_1}\right)^m
\end{aligned}
\tag{15.280}
$$

where E is the electric field, D is the dose rate, g_0 is the generation rate of electron–hole pairs, and E_0, E_1, and m are constants. All these constants can be specified in DESSIS parameter file as follows:

```
Radiation {
   g = 7.6000e+12    # [1/(rad*cm^3)]
   E0 = 0.1          # [V/cm]
   E1 = 1.3500e+06   # [V/cm]
   m = 0.9           # [1]
}
```

12.4 J-model trap equations

There is a description of different trap equations applied to radiation problems [123]. Two models formulated there are the V-model and J-model. The V-model is absolutely equivalent to the conventional trap equations that are implemented in DESSIS and described in Section 10.2 on page 15.226.

The J-model depends on the values of carrier currents (compared to the V-model where only local carrier concentrations are used). The trap equations in Section 10.2 can be rewritten for the J-model as follows:

$$-\frac{dn_\mathrm{t}}{dt} = v_\mathrm{th}^n \sigma_n N_\mathrm{Et}\left[\frac{n_1}{g_n} f_n - \tilde{n}(1-f_n)\right] - v_\mathrm{th}^p \sigma_p N_\mathrm{Et}\left[\frac{p_1}{g_p}(1-f_n) - \tilde{p} f_n\right]$$
$$\tilde{n} = \left(1-g_n^J\right)\cdot n + g_n^J \frac{|J_n|}{q v_\mathrm{th}^n} \qquad (15.281)$$
$$\tilde{p} = \left(1-g_p^J\right)\cdot p + g_p^J \frac{|J_p|}{q v_\mathrm{th}^p}$$

$$\sigma_{n,p} = \sigma_{n,p}^0\left(1 + a_1\left|\frac{E}{E_0}\right|^{p_1} + a_2\left|\frac{E}{E_0}\right|^{p_2}\right)^{p_0}, E_0 = 1\frac{V}{m} \qquad (15.282)$$

If the J-model factors (g_n^J, g_p^J) are both equal to 1, these equations will give a pure J-model as described in the literature [123]. This implementation attempts a model that provides a continuous transition between the pure V-model and pure J-model. The equations also include formulas of the electric field–dependent cross section, which follow from measurements.

All parameters can be changed in the `Traps` section of the DESSIS parameter file:

```
Traps:
{   *   XsecFormula=1: Xsec(F) = Xsec
    *   XsecFormula=2: Xsec(F) = Xsec*(1+a1*(F/1)^p1+a2*(F/1)^p2)^p0
        XsecFormula = 1 , 1                   # [1]
        Xsec = 1.0000e-15 , 1.0000e-15        # [cm^2]
        a1 = 0.0000e+00 , 0.0000e+00          # [1]
        p1 = 1 , 1                            # [1]
        a2 = 0.0000e+00 , 0.0000e+00          # [1]
        p2 = 1 , 1                            # [1]
        p0 = 1 , 1                            # [1]
        Jcoef = 0.0000e+00 , 0.0000e+00       # [1]
}
```

J-model factors can also be specified separately for each trap distribution in the input command file:

```
eJfactor = <value>, hJfactor = <value>
```

CHAPTER 13 Optical generation

13.1 Photon beam generation

DESSIS supports the simulation of photo generation using multiple vertical photon beams. Figure 15.43 illustrates the distribution of incident beam intensity in a 3D device.

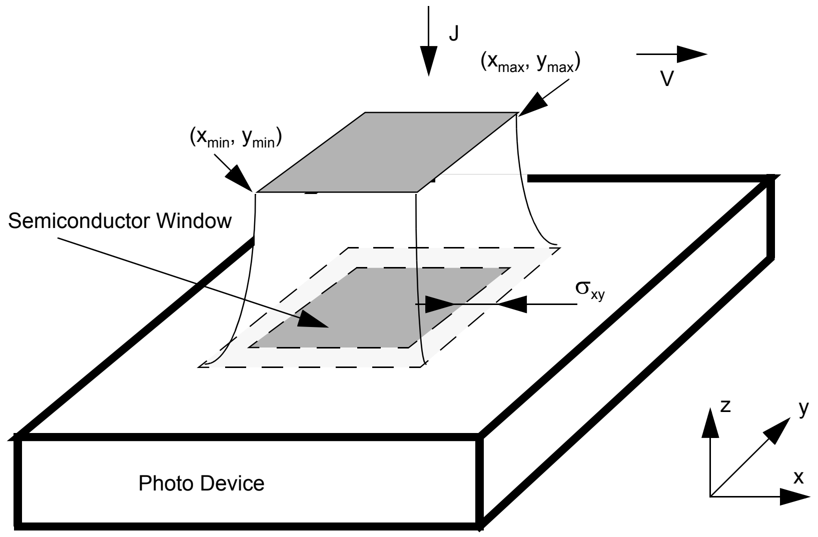


Figure 15.43   Three-dimensional distribution of incident beam intensity

J_0 denotes the incident beam intensity (number of photons that cross an area of 1 cm^2 per 1 s) at the center of the semiconductor window. (x_{min},y_{min}) and (x_{max},y_{max}) are the coordinates of opposite corners of the semiconductor window. σ_{xy} is the standard deviation of the spacial Gaussian distribution that describes the decay of the incident beam intensity outside of the defined semiconductor window. $\vec{V}$ is the velocity of the semiconductor window. Therefore, the space shape of the incident beam intensity, defined by the semiconductor window size, Gaussian decay at the edges, and velocity, can be described as a function F_{xyv}, which is equal to 1 inside the semiconductor window and decreases to zero according to a Gaussian distribution outside the window. The following are useful relations for the photo generation problem:

$$E_{ph} = \frac{hc}{\lambda} \qquad J_0 = \frac{P_0}{E_{ph}} \tag{15.283}$$

where P_0 is the incident wave power per square [W/cm^2], λ is the wavelength [cm], h is the Planck constant [J s], c is the speed of light in vacuum [cm/s], and E_{ph} is the photon energy that is approximately equal to $\frac{1.24}{\lambda[\mu m]}$ in eV. The optical generation rate along a line parallel to the z-axis for a nonuniform absorption coefficient $\alpha(\lambda, x, y, z)$ can be written as:

$$G^{opt}(z, t) = J_0 F_t(t) F_{xyv} \cdot \alpha(\lambda, z') \cdot \exp\left(-\left|\int_{z_0}^{z} \alpha(\lambda, z')dz'\right|\right) \tag{15.284}$$

where t is the time, $F_t(t)$ is the beam time behavior function that is equal to 1 for t in [t_{min},t_{max}] and shows a Gaussian distribution decay outside the interval with the standard deviation σ_t, z_0 is the coordinate of the semiconductor surface, and $\alpha(\lambda, z)$ is the absorption coefficient along the line.

Photo generation is activated by using the `OptBeam` statement in the `Physics` section. The user can specify any number of beams, for example, several beams with different wavelengths. In addition, if the `OptBeam` statement is specified in the region or material `Physics` section, the optical generation is specified to this region or material.

```
Physics{ ...
   OptBeam( ( .beam1. ) ( .beam2. ) ... ( .beamN. ) )
}
```

Table 15.97 lists the options for each beam in the `OptBeam` statement.

Table 15.97 Keyword options for OptBeam command

Keyword	Description
`WaveInt = J0` [1/cm^2/s] or `WavePower = P0` [W/cm^2]	Specifies the incident beam intensity or incident wave power. (In this case, either the wavelength or photon energy must be specified.)
`WaveLength =` λ [cm] or `WaveEnergy = Eph` [eV]	Specifies the wavelength or photon energy.
`SemAbs =` α [cm^{-1}] or `SemAbs(value =` α `)`	Both statements specify a constant value α for the absorption coefficient. This definition does not require the wave properties of the beam (wavelength or photon energy).
`SemAbs(model=Parameter)`	The absorption coefficient is computed according to the DESSIS parameter file. This is the default.
`SemAbs(model = RSS)`	Use the silicon absorption model [155].

`SemAbs(model = ODB)`
or
`SemAbs(material = "MaterialName")`
or
`SemAbs("MaterialName")`
	In this case, DESSIS takes the absorption coefficient from the table-based optical database `TableODB`, in the DESSIS parameter file.
	If a material is specified, only the data corresponding to this material is used to compute the absorption coefficient for all regions where the beam is defined.

`SemAbs(model = "<PMI Model Name>")`	Use the mentioned PMI absorption coefficient model.
`SemSurf =` z_0 [cm]	Specifies a coordinate of the semiconductor surface. If z_0 is located inside the device, the maximum of the photo generation rate ($J_0\alpha(\lambda, z_0)$) is at this coordinate, and two beams with opposite directions are applied starting from this coordinate.
`SemWindow =` (x_{min}, x_{max}) [cm] or `SemWindow = (` (x_{min}, y_{min}) (x_{max}, y_{max}) `)`	Specifies the semiconductor window for 2D or 3D cases.
`WaveXYsigma =` σ_{xy} [cm]	Standard deviation of the spatial Gaussian distribution that describes the decay of the incident beam intensity outside the defined semiconductor window `SemWind`.
`WaveTime =` (t_{min}, t_{max}) [s]	Specifies a time interval when the incident beam intensity is constant.

Table 15.97 Keyword options for OptBeam command

Keyword	Description
`WaveTsigma =` σ_t [s]	Standard deviation of the temporal of the Gaussian distribution that describes the decay of the incident beam intensity outside the time interval `WaveTime`.
`SemVelocity = (`V_x`,` V_y`)` [cm/s]	Allows a specified window to be moved perpendicular to the direction of the incident beam, therefore, simulating a scanned beam. V_x and V_y are the components of the velocity of the window. For 2D simulations, it is sufficient to define V_x only.

If the beam intensity changes rapidly in space, the accuracy of the beam intensity computations is low for coarse meshes unless special precautions are taken. Such problems can be eliminated by increasing the accuracy of the beam distribution integration over the control volume associated with each mesh vertex. Such integrations are performed by inserting small rectangular boxes inside the control volume and by executing analytic integration inside the small boxes.

To activate this additional procedure, the keyword `RecBoxIntegr` is specified with three additional parameters to control the accuracy of the procedure:

```
RecBoxIntegr(<Epsilon> <MaxNumberOfLevels> <MaxNumberOfBoxes>)
```

A specification without any parameters defaults to `RecBoxIntegr(1e-3 10 1000)`. Figure 15.44 illustrates these rectangular boxes. For each nonrectangular control volume, DESSIS provides the steps showing how to:

1. Fill with rectangles.
2. Subdivide the rectangles until the desired accuracy or maximum refinement levels a0 re reached.

The procedure of inserting small rectangular boxes inside the control volume is also used to compute integrals:

$$\int_{z_0}^{z} \alpha(\lambda, z')dz' \tag{15.285}$$

which are used in the photo generation rate. Therefore, if the user requires such integration, the keyword `RecBoxIntegr` must also be specified.

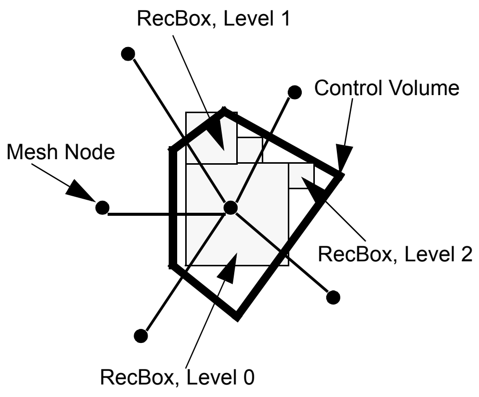


Figure 15.44 Example of rectangular boxes without parameter defaults

NOTE For many cases, the absorption coefficient is constant over all active regions of the device, and the photo generation rate can be computed by the simpler expression $J_0 \cdot \alpha \cdot \exp(-\alpha|z-z_0|)$. In this case, it is not necessary to specify the keyword. DESSIS uses local values of the absorption coefficient.

13.2 Absorption models

There are several options to define the absorption coefficient for the photon beam generation:

- A constant value is in the DESSIS input file.
- A dependence on the photon energy, mole fraction, and so on is in the DESSIS parameter file.
- A table-based optical property that depends on photon energy can be entered in the DESSIS parameter file.
- An absorption coefficient model for indirect band gap material [155].
- A new expression is created for the absorption coefficient using the PMI.

NOTE The temperature dependence of the DESSIS parameter file based absorption coefficient models, both the default model and the absorption coefficient model [155], is dependent on the global device temperature due to convergence reasons. Local temperature–dependent models can be created by using the PMI.

13.2.1 Default absorption model from DESSIS parameter file

The absorption coefficient has two available models that can be selected in the DESSIS parameter file. The simpler one is:

$$\alpha(E_{ph}) = \begin{cases} \alpha_1 \exp((E_{ph}-E_1)/E_2), E_{ph} < E_1 \\ \alpha_1 + \alpha_2((E_{ph}-E_1)/E_2)^p, E_{ph} \geq E_1 \end{cases} \tag{15.286}$$

where E_{ph} is the photon energy, and α_1, α_2 [cm^{-1}], E_1, E_2 [eV] and p are the model parameters that can be specified in the parameter file:

```
Absorption
   { Formula = 1
   A1 = <value>
   A2 = <value>
   E1 = <value>
   E2 = <value>
   p = <value>
}
```

Another model [139] for mole-dependent $Hg_{1-x}Cd_xTe$ is generalized as:

$$E_T = E_0 + \frac{T+T_0}{\sigma}\ln\frac{\alpha_T}{\alpha_0}$$

$$\alpha(E_{ph}, T) = \begin{cases} \alpha_0 \exp\left(\frac{\sigma(E_{ph}-E_0)}{T+T_0}\right), E_{ph} < E_T \\ \alpha_T\left(\frac{2\sigma}{T+T_0}\left(E_{ph}-E_0-\frac{T+T_0}{\sigma}\left(\ln\frac{\alpha_T}{\alpha_0}-0.5\right)\right)\right)^{0.5}, E_{ph} \geq E_T \end{cases} \tag{15.287}$$

where α_T, α_0 [cm^{-1}], E_0 [eV], T_0 [K], and σ [K/eV] are model parameters that can be specified in the parameter file:

```
Absorption
   { Formula = 2
   AT = <value>
   A0 = <value>
   E0 = <value>
   T0 = <value>
   S = <value>
}
```

All parameters of both models can have a mole dependence for mole-dependent materials using the standard DESSIS technique with linear interpolation on specified mole intervals.

However, for the material $Hg_{1-x}Cd_xTe$, [139] provides the following mole dependencies:

$$\begin{aligned} \alpha_0 &= \exp(-18.88 + 53.61x) \\ \alpha_T &= 100 + 5000x \\ \sigma &= 3.267\times10^4(1+x) \\ E_0 &= -0.3424 + 1.838x \\ T_0 &= 81 \end{aligned} \tag{15.288}$$

where x denotes the CdTe mole fraction. The parameter α_0 has an exponential dependence that cannot be described by the linear interpolation used for mole-dependent parameters in DESSIS. To activate the above expressions, the following must be present in the parameter file:

```
Absorption { Formula = 2 HgCdTe }
```

13.2.2 Table-based optical properties of materials in DESSIS parameter file

Table entry of real refractive index, n, and extinction coefficient, k, versus wavelength. Absorption coefficient is computed from k by $\alpha(\lambda) = 2\frac{\omega}{c}k$.

DESSIS has the tabled values of these optical properties for silicon, silicon dioxide, aluminum, gold, silver, platinum, silicon nitride, tungsten, and air (gas). For other materials, the `TableODB` section can be inserted in the DESSIS parameter file, and the user can create a table for a material or a region.

The first element in a row is the wavelength, followed by n and k. The row is terminated by a semicolon. There can be any number of rows greater than 3, but not less, for a cubic spline to be formed from the table entries.

```
TableODB
{ *Table format of the Optik DataBase
*WAVELENGTH      n          k
       0.051     0.804      0.322;
       0.053     0.811      0.366;
       0.055     0.822      0.408;
       0.056     0.829      0.43;
       0.058     0.843      0.47;
}
```

13.2.3 Absorption coefficient model

An absorption model [155] is implemented in DESSIS for silicon and materials with similar band gap structure. The model provides an absorption coefficient model and fits absorption data of silicon supplied by NASA, by varying temperature and photon energy. The model is switched on by inserting the statement `SemAbsorption` in the appropriate `Physics` section.

NOTE The placement of this statement above is outside either `RayTrace` or `OptBeam` where it has traditionally resided.

The resulting equation with absorption coefficient α is:

$$\alpha(T) = \sum_{i=1}^{2}\sum_{j=1}^{2} C_i A_j \left[\frac{\{hf - E_{gj}(T) + E_{pi}\}^2}{e^{\frac{-E_{pi}}{kT}} - 1} + \frac{\{hf - E_{gj}(T) - E_{pi}\}^2}{1 - e^{\frac{-E_{pi}}{kT}}}\right] + A_d[hf - E_{gd}(T)]^{\frac{1}{2}} \qquad (15.289)$$

where $E_g(T) = E_g(0) - [\beta T^2/(T+\gamma)]$ and T is the temperature [K]. Table 15.98 lists the default parameter values for silicon used in DESSIS as published [155].

Table 15.98 Default parameter values of silicon

Quantity	Value	Comment
Eg1(0)	1.1557 [eV]	Indirect gap
Eg2(0)	2.5 [eV]	Indirect gap
Egd(0)	3.2 [eV]	Direct allowed gap
Ep1	1.827e-2 [eV]	TA, Theta = 212 K
Ep2	5.773e-2 [eV]	T0, Theta = 670 K
C1	5.5	–
C2	4.0	–
A1	3.231e2 [cm^{-1}/eV^2]	–
A2	7.237e3 [cm^{-1}/eV^2]	–
Ad	1.052e6 [cm^{-1}/eV^2]	–
Beta	7.021e-4 [eV/K]	–
Gamma	1108 [K]	–

Further, these parameter values can be changed for some materials with band gap structures similar to silicon or for silicon itself, by using the following (appropriate for DESSIS parameter files) and replacing the parameters with desired values:

```
RSSAbsorption
{ * K. Rajkanan, R. Singh, and J. Shewchun,
  * Absorption Coefficient of Silicon for Solar Cell Calculations
  * Solid-State Electronics 1979 Vol 22. pp793-795
          Egone0 = 1.1557      # [eV]
          Egtwo0 = 2.5         # [eV]
          Egd0 = 3.2           # [eV]
          Ep1 = 0.01827        # [eV]
          Ep2 = 0.05773        # [eV]
          C1 = 5.5             # [1]
          C2 = 4               # [1]
          A1 = 3.2310e+02      # [cm^-1 eV-2]
          A2 = 7.2370e+03      # [cm^-1 eV-2]
          Ad = 1.0520e+06      # [cm^-1 eV-2]
          RssBeta = 7.237e-4   # [eV/K]
          RssGamma = 1108      # [K]
}
```

If the `RSSAbsorption` model is toggled in `Physics` and if the above `RSSAbsorption` model parameters are not specified in the DESSIS parameter file, then the parameter values of silicon are used.

The mole fraction dependence of the `RSSAbsorption` model is identical to that of the default DESSIS parameter-based absorption model (see Section 13.2.1 on page 15.246).

The absorption model specification in the `Physics` section overwrites that in the `OptBeam` or `RayTrace` section.

13.3 Optical generation by raytracing

A plane wave can be partitioned and each partition is represented into a one-dimensional ray of light. Raytracing can approximate the behavior of a plane wave on a device by following such rays.

NOTE If the partition is too sparse compared to the element size of the device being simulated, the resulting approximation will be unsatisfactory.

13.3.1 Overview

DESSIS supports simulation of photo generation by raytracing, which allows for the simulation of photo generation on devices of more complicated geometries than the photon beam generation described previously.

The optical generation along a ray, when the propagation is thought to be in the z-axis for a nonuniform absorption coefficient $\alpha(\lambda, x, y, z)$, can be written as for photon beam generation:

$$G^{opt}(z,t) = J(x,y,z_o)\alpha(\lambda,z)\exp\left(-\left|\int_{z_0}^{z}\alpha(\lambda,z)dz\right|\right) \tag{15.290}$$

where $J(x, y, z_o)$ is the beam spatial variation of intensity over a window where rays enter the device, and z_o is the position along the ray where absorption begins.

Photo generation by raytracing, which will be referred to as *raytracing* from this point, is activated by the `RayTrace` statement in the `Physics` section. A `RayTrace` statement defines a set of beams that will be represented by rays and several beams with different properties (such as different intensities, wavelengths, and absorptions) can be defined:

```
Physics {...
   RayTrace((RayBeam1) (RayBeam2)... (RayBeamN))
}
```

Table 15.99 lists the options for each beam in the `RayTrace` statement.

Table 15.99 Keyword options for RayTrace command

Keyword	Description
`WaveInt = J0` [1/cm^2/s] or `WavePower = P0` [W/cm^2]	Specifies the incident beam intensity or incident wave power. (In this case, either the wavelength or photon energy must be specified.)
`WaveLength =` λ [cm] or `WaveEnergy = Eph` [eV]	Specifies the wavelength or photon energy.
`WaveTime =` (t_{min}, t_{max}) [s]	Specifies a time interval when the incident beam intensity is constant.
`WaveTsigma =` σ_t [s]	Standard deviation of the temporal of the Gaussian distribution that describes the decay of the incident beam intensity outside of the time interval `WaveTime`.
`WaveDirection = v`	Direction that the rays will travel is specified by the vector `v`.
`SemAbs =` α [cm^{-1}] or `SemAbs(value =` α `)`	Both statements specify a constant value α for the absorption coefficient. This definition does not require the wave properties of the beam (wavelength or photon energy).
`SemAbs(model=Parameter)`	The absorption coefficient is computed according to the DESSIS parameter file. This is the default.
`SemAbs(model = RSS)`	Use the silicon absorption model [155].

`SemAbs(model = ODB)`
or
`SemAbs(material = "MaterialName")`
or
`SemAbs("MaterialName")`
	In this case, DESSIS takes the absorption coefficient from table-based optical database `TableODB`, in the DESSIS parameter file. If a material is specified, only the data corresponding to this material is used to compute the absorption coefficient for all regions where the beam is defined.

`SemAbs(model = "<PMI model name>")`	Uses the noted PMI absorption coefficient model.
`RefractiveIndex(value =` n `)`	Specifies a constant value n for refractive index.
`Refractive Index (model=Parameter)`	The refractive index is computed according to the DESSIS parameter file.
`RefractiveIndex (model=ODB)`	Takes the value of the refractive index from the table-based optical database `TableODB`, in DESSIS parameter file.

`RefractiveIndex(model =`
`"<PMI model name>")`
	Uses the mentioned PMI refractive index model.

`RefractiveIndex (model = ODB)`	Takes the value of the refractive index from the optical database file `optikdata`.

Table 15.99 Keyword options for RayTrace command

Keyword	Description
`MinIntensity = r`	One of two termination conditions for tracing rays. If the intensity of a ray becomes less than `r` times the original intensity, DESSIS does not trace the ray further.
`DepthLimit = d`	The other termination condition for tracing rays. If a ray passes through more than `d` material boundaries, DESSIS does not trace the ray further.
`CircularWindow{...}` or `RectangularWindow{...}`	Statement group that defines various properties of the window that allows rays through. Table 15.100 lists the contents of these statements.
`Print`	This statement creates a `.grd` file with information about the paths that the rays take. The resulting plot file name is "`<dessis plot file>_ray.grd`" if the plot file is specified. If no plot file is specified, then it is assigned a default name of "`Raytrace_ray.grd`".

Table 15.100 Keyword options for CircularWindow{ . . .} or RectangularWindow{ . . .}

Keyword	Description
`CellSize = s` [μm]	Determines the spacing between rays. The definition of `CellSize` varies, depending on the dimension of the simulation and the window type.
`WindowCenter = C` [μm, μm, μm]	Position of the center of the window.
`WindowDirection = D`	Direction that the window faces.
`WindowSize = S` or `WindowRadius = R` or `WindowHeight = H` and `WindowWidth = W` [μm]	Specifies the size of window. `WindowSize` for any 2D structure. `WindowRadius` for `CircularWindow` of any dimensionality. `WindowHeight` and `WindowWidth` for `RectangularWindow` of a 3D device.
`intvalue1` and `intvalue2`	Specifies the value for the spatially varying intensity spline function.
`intposition1` and `intposition2` or `intpositionnormalized1` and `intpositionnormalized2`	Specifies the position of the values specified in `intvalue1` and `intvalue2` for the spatially varying intensity spline function.
`Polarization(...)`	Inserts the initial polarization description of rays. Table 15.101 lists the options for `Polarization`.

Table 15.101 Keyword options for Polarization

Keyword	Description
`Axis1`	Relative length of either major or minor axis.
`Axis2`	Relative length of the other axis direction from `Axis1`.
`Axis1Dir`	Direction of `Axis1`. `Axis2` is the perpendicular to `Axis1` and direction of the ray.

13.3.2 Snell's law, and refraction and reflection intensities

When a ray passes through a material boundary, it splits into two rays. The angles involved in DESSIS raytracing are governed by Snell's law, shown in Figure 15.45.

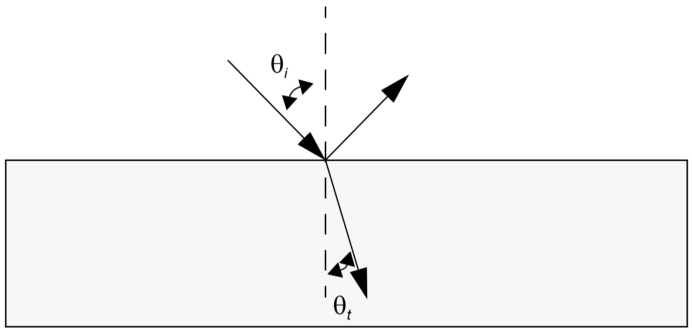


Figure 15.45 Snell's law

$$n_1\sin(\theta_i) = n_2\sin(\theta_t) \tag{15.291}$$

The following formulas describe the intensity of the refracted electromagnetic wave on a plane surface between two media:

$$T_{\text{perp}} = \frac{\sin(2\theta_i)\sin(2\theta_t)}{\sin^2(\theta_i+\theta_t)} \tag{15.292}$$

$$T_{\text{par}} = \frac{\sin(2\theta_i)\sin(2\theta_t)}{\sin^2(\theta_i+\theta_t)\cos^2(\theta_i+\theta_t)} \tag{15.293}$$

The above formulas decompose the planar wave into a polarization component that is perpendicular to the plane of incidence, the plane that the ray propagation direction and the vector normal to the plane surface lie on, and a component parallel to the plane of incidence. Reflection coefficients can be computed by subtracting the transmission coefficients from 1:

$$1 - T_{\text{par}} = R_{\text{par}} \tag{15.294}$$

$$1 - T_{\text{perp}} = R_{\text{perp}} \tag{15.295}$$

The possibility of absorption of a photon on the interface is disregarded.

Reflection coefficients of metals

An incident ray on metal is assumed to be reflected completely.

13.3.3 Polarization

Polarization information about a ray is stored as pairs of points that form an ellipse. This is not the elliptical polarization conventionally thought of, but a point on the ellipse that represents the intensity of a component of the ray polarized in that direction.

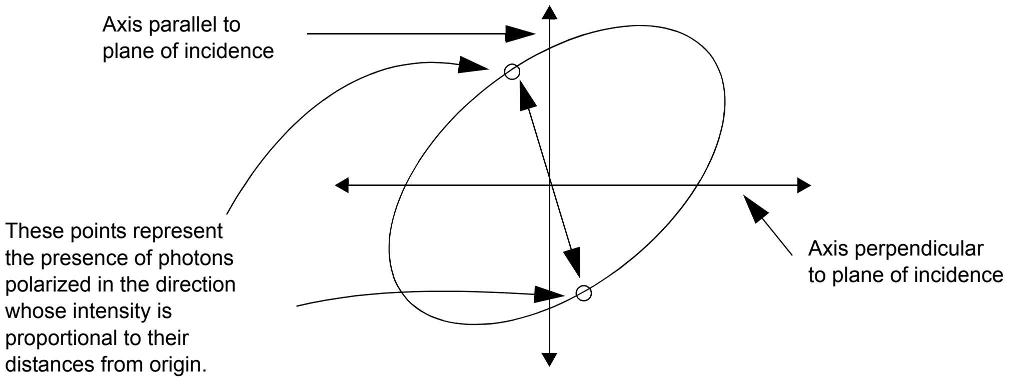

Axis parallel to plane of incidence
These points represent the presence of photons polarized in the direction whose intensity is proportional to their distances from origin.
Axis perpendicular to plane of incidence

Figure 15.46 Polarization ellipse

The ray represents a wave that is considered to be ‘in phase.’ The seemingly out-of-phase components, for example, the two points noted in Figure 15.46, are present to make operations on the ellipse itself more convenient. The ellipse arises naturally when considering that a wave of unpolarized photons, which has equal intensity components in all directions, would be represented as a circle. If this circle is transformed by the above transmission and reflection coefficients, according to the parallel and perpendicular decomposition, the result is an ellipse.

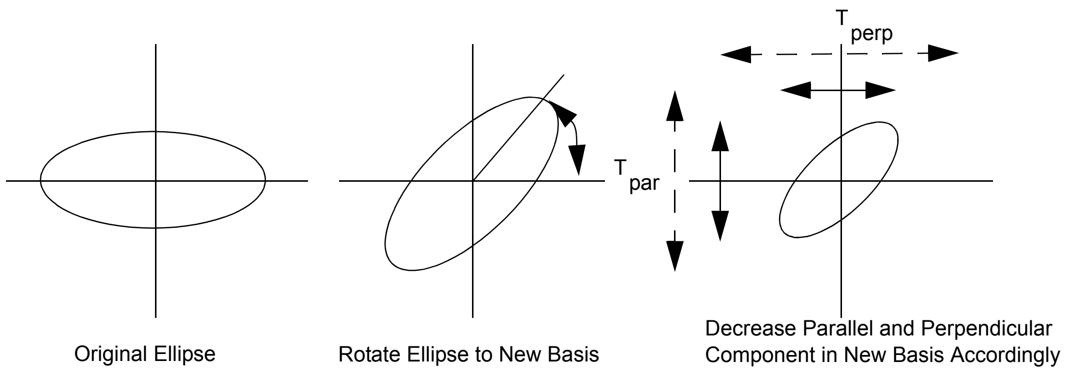

T perp
T par
Original Ellipse
Rotate Ellipse to New Basis
Decrease Parallel and Perpendicular Component in New Basis Accordingly

Figure 15.47 Operations performed on polarization ellipse of a ray upon its passing through a material boundary

Two things may happen to the polarization ellipse in Figure 15.47 when the ray corresponding to it passes through a material boundary:

- If the new plane of incidence of the material boundary is different from the one on which the current axes of polarization are defined, the ellipse requires a change of basis into the vectors parallel and perpendicular to the new plane of incidence.
- Polarization components parallel and perpendicular to the plane of incidence are multiplied by T_{xxx} if the ray is a refracted portion of the original ray, or R_{xxx} if the ray is a reflected portion.

Initial polarization of a ray can be specified by the `Polarization` statement, which must have the following format:

```
RayTrace(...
   Polarization(
      axis1 = a
      axis2 = b
      axis1dir = v
   )
)
```

where `a` and `b` are real numbers, and `v` is a vector in three-dimensional space. The ratio between `a` and `b` is important, not their absolute values. The vector `v` specified in the input does not need to be orthogonal to the direction, it only needs to be nonidentical to it. For the purpose of polarization, a projected value `v` from the plane orthogonal to the ray direction is used.

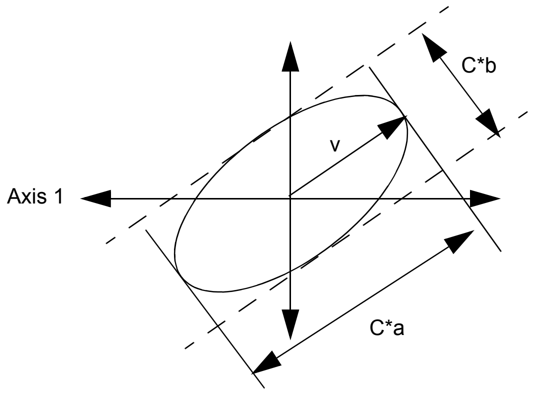


Figure 15.48 Polarization syntax

Although in Figure 15.48, axis 1 is depicted as the major axis, this does not need to be the case. C is a constant that ensures the intensity represented by the ellipse has a correct value.

13.3.4 Absorption models

The absorption models for raytrace generation are identical to those of photon beam generation (see Section 13.2 on page 15.246).

13.3.5 Refractive index model

There are several options to define the refractive index for ray trace generation:

- A constant value can be defined in the DESSIS input file.
- A dependence on the mole fraction and temperature is in the DESSIS parameter file.
- A table-based optical property that depends on photon energy can be entered in the DESSIS parameter file (see Section 13.2.2 on page 15.247).
- A new expression is created for the refractive index using the PMI.

This section discusses the second option. For the refractive index of a material at 300 K, the refractive index n_{300} is determined by:

$$n(T) = n_{300}(1 + \alpha \times (T - T_{par})) \tag{15.296}$$

where α and T_{par} are parameters specified in the DESSIS parameter file.

```
RefractiveIndex {
   * refractiveindex() = refractiveindex * (1 + aplpha * (T-Tpar))
   alpha = 2.0000e-04    # [1/K]
   Tpar = 3.0000e+02     # [K]
}
```

Mole-fraction dependence is performed in the standard DESSIS polynomial spline fashion.

13.3.6 Intensity

A point on the ellipse represents a component in the photon beam that is polarized in that direction. The distance of the point from the origin is the intensity of that component. The total intensity of a ray is an integral of the distance of each point from the center of the ellipse.

For example, let the initial polarization ellipse have minor and major axes of 2a and 2b, respectively:

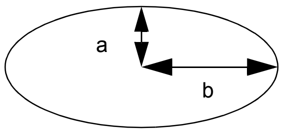


Figure 15.49 Polarization ellipse

$$I = I_o \int_0^{2\pi} \sqrt{a^2 \sin^2\theta + b^2 \cos^2\theta}\, d\theta \tag{15.297}$$

The intensity is described by (Eq. 15.297), where I_o is a constant of proportionality. The above integral does not have a closed form solution.

The second formula of Ramanujan is used, which approximates the above integral by:

$$I = I_0 \pi (a + b)\left[1 + \frac{3h}{10 + \sqrt{4 - 3h}}\right] \tag{15.298}$$

where $h = \left(\frac{a-b}{a+b}\right)^2$.

13.3.7 Window of ray

In DESSIS raytracing, rays of photon come through some type of window. The window can be either circular or rectangular, although for 2D structures, it makes no difference. The description of a window in DESSIS raytracing is designed to be as flexible as possible.

13.3.7.1 2D device and its window description

For a 2D device, a window is one-dimensional. The only properties of the window that must be defined are:

- Where it is (the center of the window)
- Its size
- Which direction it faces

A window size can be described by specifying `WindowRadius` or `WindowSize`.

13.3.7.2 Example A: 2D device ray window

```
RayTrace(...
   CircularWindow(
      windowcenter = (10,12,0)
      WindowDirection = (0,-1,0)
      CellSize = 0.1
      WindowRadius = 5
   )
)
```

`windowcenter` notes the position of the center of the window. `WindowDirection` notes the direction that the window's flat side is facing, which can be different from the actual wave propagation direction. The size of a 2D device raytracing window is specified by either `WindowSize` or `WindowRadius`. `WindowSize` works for both rectangular and circular windows, and `WindowRadius` works only for the circular window type. This complexity arose from a requirement to have the same syntax for 2D and 3D circular windows, although for rectangular windows, having the same syntax for all dimensions was not possible.

NOTE By inputting the vector position for 2D, the value of the **z** component must be zero.

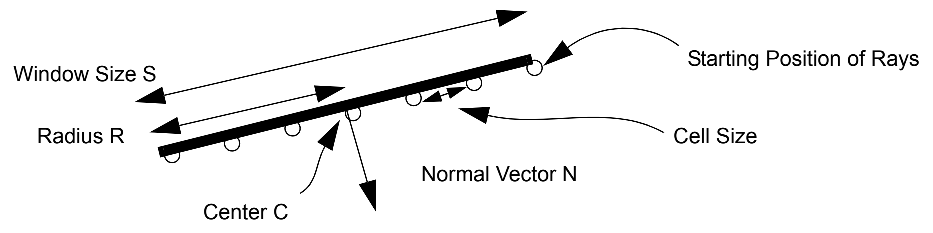


Figure 15.50 Window for 2D devices

The code of this example results in an identical window to example B.

13.3.7.3 Example B: 2D device ray window

```
RayTrace(...
   CircularWindow(
      windowcenter = (10,12,0)
      WindowDirection = (0,-1,0)
      CellSize = 0.1
      WindowSize = 10   # <-- changed here from Ex. A
   )
)
```

Both examples A and B result in an identical window to that generated by example C.

13.3.7.4 Example C: 2D device ray window

```
RayTrace(...
       rectangularwindow(   # <-- changed here from Ex. B
          windowcenter = (10,12,0)
          WindowDirection = (0,-1,0)
          CellSize = 0.1
          WindowSize = 10
       )
)
```

Within the window, spatial distribution of the ray is uniform. The spacing is determined by the `CellSize` parameter, which specifies the distance [µm] between ray starting points. For a 2D device, the parameter `NumberOfRays` is used to place rays.

13.3.7.5 Example D: Number of rays

```
RayTrace(...
       CircularWindow(
          windowcenter = (10,12,0)
          WindowDirection = (0,-1,0)
          NumberOfRays = 100   # <-- changed here
          WindowSize = 10
       )
)
```

If the specified number of rays is N, then N rays are placed uniformly over the window.

13.3.7.6 3D devices and their windows: CircularWindow

The two window types, circular and rectangular, have some meaning for 3D devices. `CircularWindow` is a window that is circular. Basic syntax is identical to the instance of a 2D circular window described in Section 13.3.7.2 on page 15.256 (but not the others).

13.3.7.7 Example A: 3D device ray window

```
RayTrace(...
       CircularWindow(
          windowcenter = (10,12,0)
          WindowDirection = (0,-1,0)
          CellSize = 0.1
          WindowRadius = 5
       )
)
```

The meanings of all keywords are identical except `CellSize`, which is now the length of an equilateral triangle that makes up the lattice of ray starting points.

The alignment of the lattice is controlled by the keyword `WindowOrientation`.

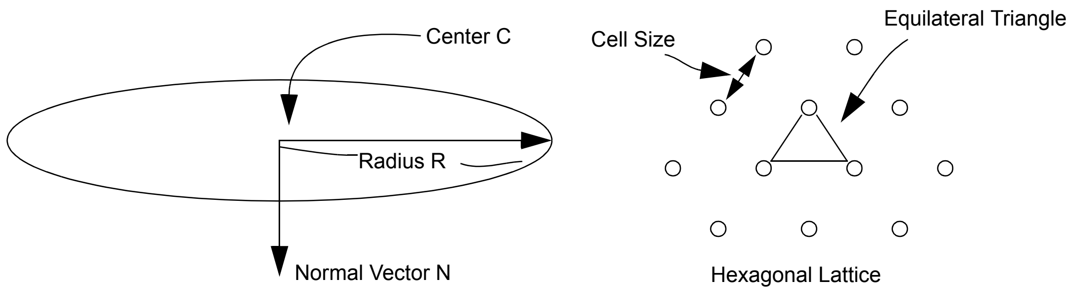


Figure 15.51 Circular window for 3D device and its lattice of rays

13.3.7.8 Example B: 3D device ray window

```
RayTrace(...
       CircularWindow(
          windowcenter = (10,12,0)
          WindowDirection = (0,-1,0)
          CellSize = 0.1
          WindowRadius = 5
          WindowOrientation = (0,0,1)
       )
)
```

This code aligns one side of the equilateral triangle to the direction of (0,0,1). `WindowCenter` and `WindowDirection` have the same meaning as in 2D device windows.

13.3.7.9 3D devices and their windows: RectangularWindow

Unlike the circular window, which requires only one length description (namely radius), a rectangle requires two: height and width. Orienting this rectangle is performed using the `WindowOrientation` vector. This vector specifies the upward and downward direction, or the direction of the side whose length is defined by the `WindowHeight` field.

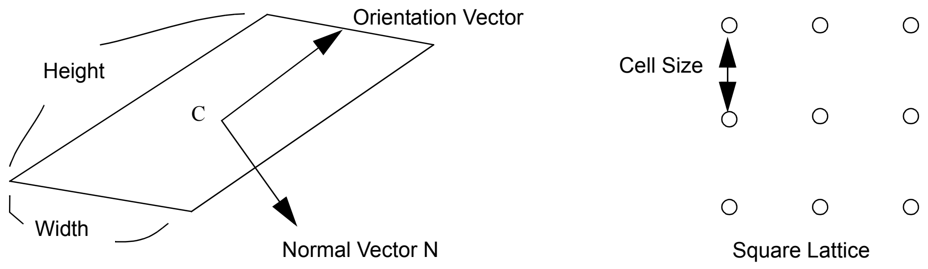


Figure 15.52 Rectangular window for 3D device and its lattice of rays

13.3.7.10 Example A: 3D device ray window

```
RayTrace(...
       RectangularWindow(
          windowcenter = (10,12,0)
          WindowDirection = (0,-1,0)
          CellSize = 0.1
```

```
            WindowHeight = 5
            WindowWidth = 10
            WindowOrientation = (0,0,1)
        )
)
```

If `WindowOrientation` is not specified or the specified orientation is identical to `WindowDirection`, a vector perpendicular to `WindowDirection` is assigned to it. `CellSize` for a 3D device rectangular window is the length of a side of the square that makes up the lattice of ray starting positions. Again, `WindowCenter` and `WindowDirection` have the same meaning as in 2D device windows.

13.3.8 Spatial distribution of intensity

In DESSIS, the intensity of rays within a given window can vary by using the keywords `intvalue1`, `intvalue2`, and `intposition1`, `intposition2`, or `intPositionNormalized1`, `intPositionNormalized2` in the `CircularWindow` or `RectangularWindow` statement group.

13.3.8.1 Method 1: Linear splines

`intvalueX` defines a spline, and represents ‘intensity value.’

13.3.8.2 Ray window with spatially varying intensity

```
RayTrace(...
        CircularWindow(
            windowcenter = (10,12,0)
            WindowDirection = (0,-1,0)
            CellSize = 0.1
            WindowRadius = 5
            intvalue1 = [1, 1, 0.5, 0.3, 0.3, 0]
        )
)
```

The `intvalue1` statement in Figure 15.53 defines a linear spline. Let the vector specified in `intvalue1` be `v = [v1, v2, v3, ..., vn]`.

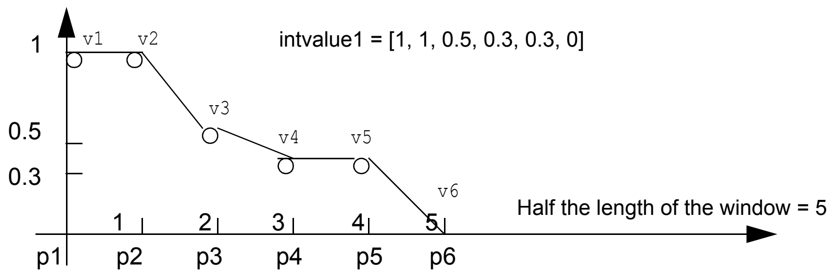


Figure 15.53 Linear spline from example input

For a given window, a set of points `p = [p1, p2, p3, ..., pn]` that equidistantly partition one half of the window can be defined.

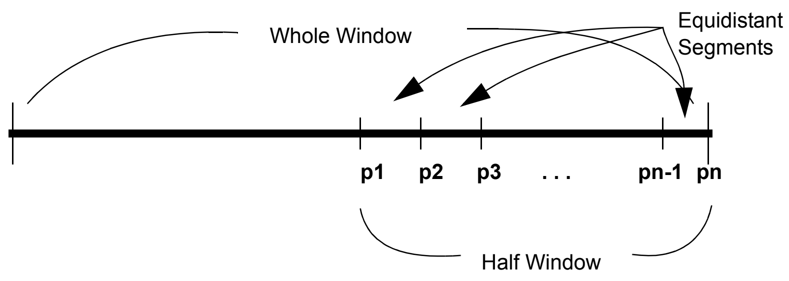


Figure 15.54 Half window and its partition

A spline is defined over this partitioned half window. Let this spline function be **f(d)**, where **d** is the distance from the center of the window.

13.3.8.3 2D device window intensity

In a 2D device, the spline in Figure 15.53 on page 15.259 is mirrored about the y-axis, and stretched or shrunk to fit the window.

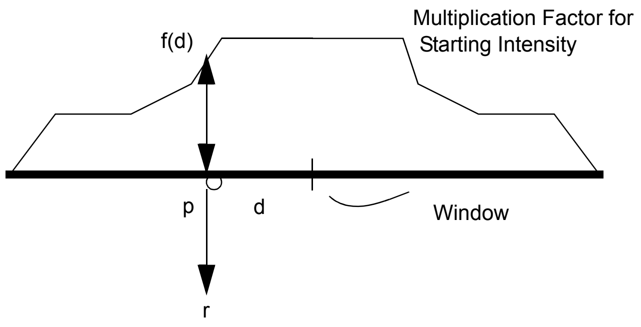


Figure 15.55 Multiplication for starting intensity

Stating this more formally, for a given ray `r`, starting in the window position `p`, distance `d` away from the center of the window, the starting intensity `I`, specified by the `waveint` command, is internally modified for the ray `r` to be `f(d)*I`. An example showing how to insert a Gaussian intensity profile follows.

13.3.8.4 Example A: Gaussian intensity

```
RayTrace(...
       CircularWindow(
          windowcenter = (10,12,0)
          WindowDirection = (0,-1,0)
          CellSize = 0.1
          WindowRadius = 5
          intvalue1 = [ 1.0000    0.9610    0.8529    0.6990
                        0.5291    0.3698    0.2387    0.1423
                        0.0784    0.0398    0.0187    0.0000]
       )
)
```

If the device being simulated is a 2D device, the above code creates a window of length 10 µm, centered at position (10, 12), with the intensity proportional to Figure 15.56, which is the plot of the string of numbers in the `intvalue1` vector in Section 13.3.8.4 on page 15.260.

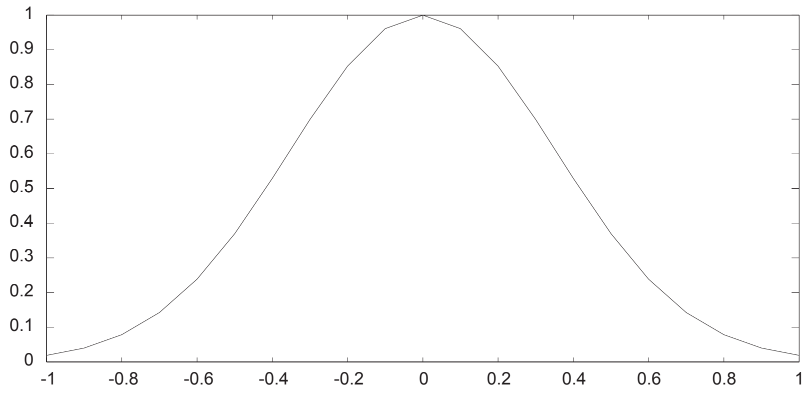


Figure 15.56 Gaussian intensity

13.3.8.5 3D circular window intensity

For a circular window, use the same spline function **f(d),** still defined in terms of distance from the center. A starting point of a ray **p**, distance **d** away from the center, will still be **f(d)*I**. The resulting intensity will appear as in Figure 15.57 on page 15.262 for the following input code.

13.3.8.6 Example B: Gaussian intensity

```
RayTrace(...
       CircularWindow(
       windowcenter = (10, 12,0)
       WindowDirection = (0,-1,0)
       CellSize = 0.1
       WindowRadius = 5
       intvalue1 = [ 1.0000     0.9610     0.8529     0.6990
                     0.5291     0.3698     0.2387     0.1423
                     0.0784     0.0398     0.0187     0.0000]
       )
)
```

NOTE No rays are defined outside the window, which in the case of Figure 15.57 would be a circle of radius 5.

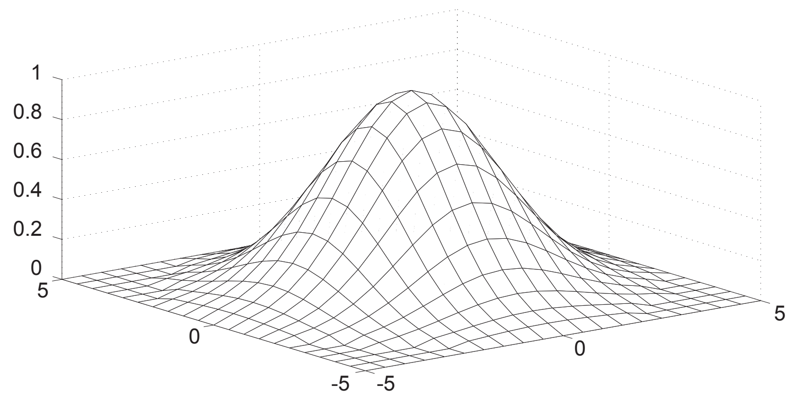


Figure 15.57 Intensity in a 3D device circular window with intval1

13.3.8.7 3D device rectangular window

With the same method, adding the `intvalue1` vector, for a rectangular window yields a different result.

13.3.8.8 Gaussian intensity on one side

```
RayTrace(...
       RectangularWindow(
          windowcenter = (10,12,0)
          WindowDirection = (0,-1,0)
          CellSize = 0.1
          WindowHeight = 5
          WindowWidth = 3
          WindowOrientation = (0,0,1)
          intvalue1 = [ 1.0000     0.9610     0.8529     0.6990
                        0.5291     0.3698     0.2387     0.1423
                        0.0784     0.0398     0.0187     0.0000]
       )
)
```

This input code will result in the starting profile shown in Figure 15.58.

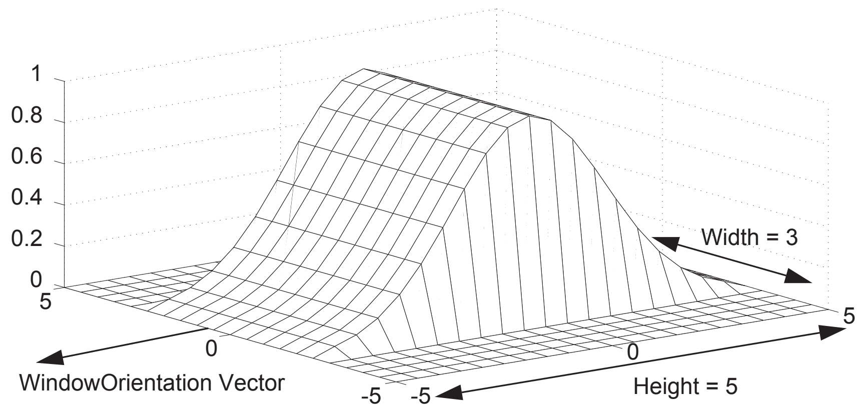


Figure 15.58 3D rectangular window with only intval1 specified

Viewed from the side, this is identical to that of the 2D device window intensity. The `intvalue2` vector specifies the intensity description along the 'width' direction.

13.3.8.9 3D rectangular window

```
RayTrace(...
       RectangularWindow(
          windowcenter = (10,12,0)
          WindowDirection = (0,-1,0)
          CellSize = 0.1
          WindowHeight = 5
          WindowWidth = 3
          WindowOrientation = (0,0,1)
          intvalue1 = [ 1.0000    0.9610    0.8529    0.6990
                        0.5291    0.3698    0.2387    0.1423
                        0.0784    0.0398    0.0187    0.0000]
          intvalue2 = [ 1.0000    0.9610    0.8529    0.6990
                        0.5291    0.3698    0.2387    0.1423
                        0.0784    0.0398    0.0187    0.0000]
       )
)
```

Identical to values in the `intvalue1` string of numbers, the resulting spline function from `intvalue2` is similar in shape to that resulting from `intvalue1`, but thinner by a ratio of 3:5, the ratio of the window sizes.

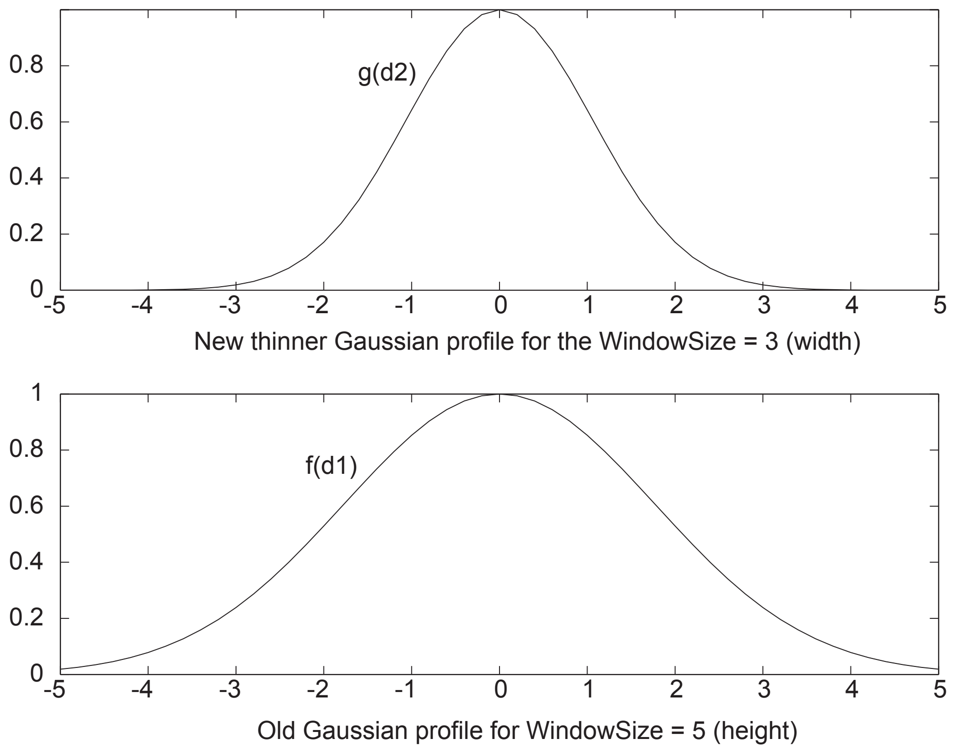
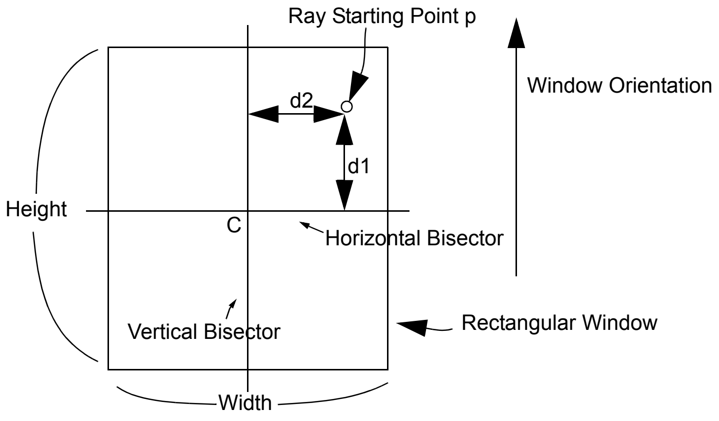


Figure 15.59   Old Gaussian profile for WindowSize = 5 (height)

Let **f(d1)** continue to represent the spline function formed from `intvalue1`, and let **g(d2)** represent the spline function formed from `intvalue2`. For a given ray starting point `p` in the window, `d1` is the distance from the point position to the 'horizontal' line bisecting the window. Whereas, `d2` is the distance from the point position to the 'vertical' line bisecting the window.

Horizontal and vertical are defined by their relation to the `WindowOrientation` vector:



As before, if **I** was specified in the `waveint` input statement, the intensity of a ray starting at point `p` is internally multiplied by **f(d1)** and **g(d2)** to be **f(d1)*g(d2)*I**. Therefore, for the example code given in Section 13.3.8.9 on page 15.263, the following intensity pattern will arise as shown in Figure 15.60.

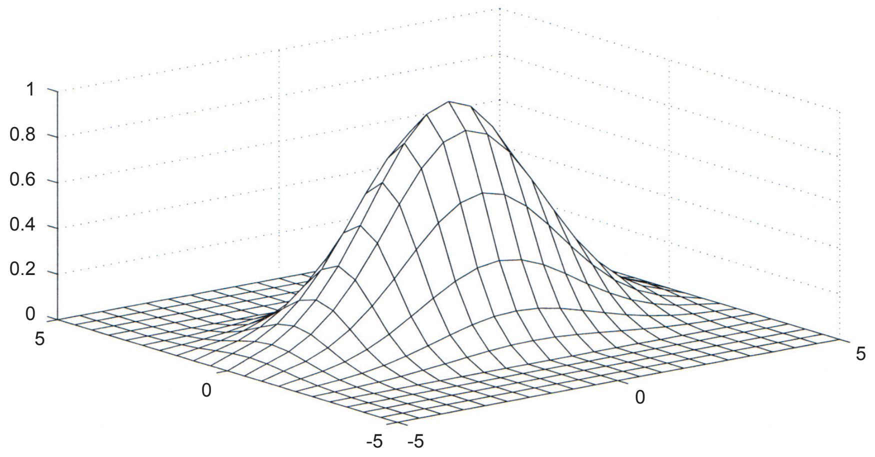


Figure 15.60 3D device rectangular window intensity pattern

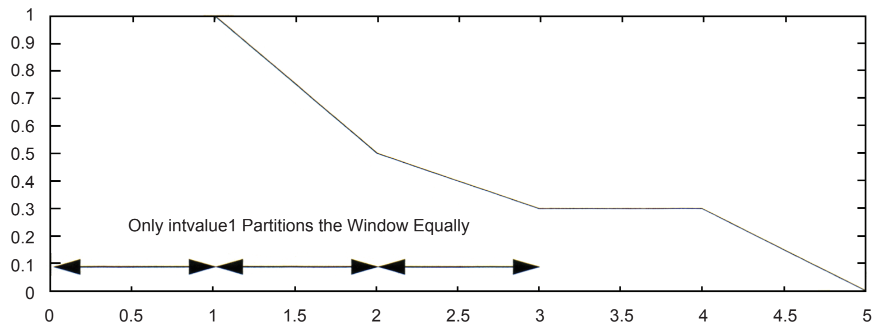
13.3.8.10 More flexible spline description

Simply inputting `intvalueX` vector makes a spline with uniform spacing of data points. While this works reasonably well with a smoothly varying function like the Gaussian distribution function, to describe a sharply varying function adequately would require too much redundant information. Take the earlier example.

13.3.8.11 Ray window with spatially varying intensity revisits I

```
RayTrace(...
       CircularWindow(
          windowcenter = (10,12,0)
          WindowDirection = (0,-1,0)
          CellSize = 0.1
          WindowRadius = 5
          intvalue1 = [1, 1, 0.5, 0.3, 0.3, 0]
       )
)
```

This example results in the following spline function:



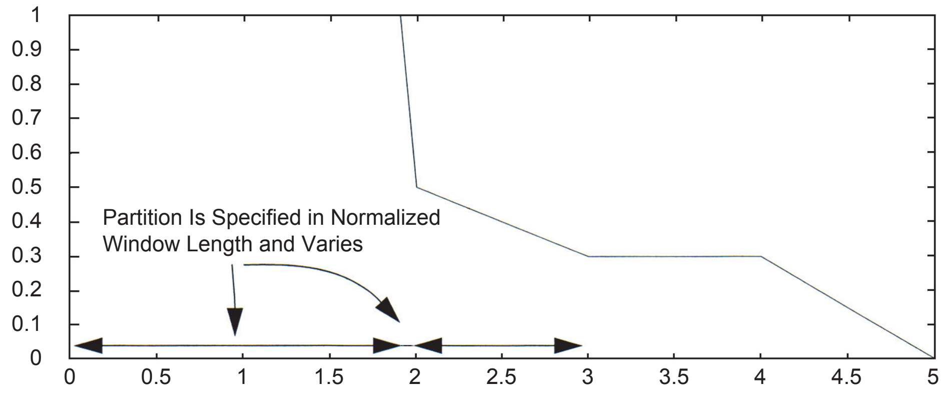
However, for a sharper drop from value 1 to 0.5 at d = 2, use the following code.

13.3.8.12 Ray window with spatially varying intensity revisits II

```
RayTrace(...
       CircularWindow(
          windowcenter = (10,12,0)
          WindowDirection = (0,-1,0)
          CellSize = 0.1
          WindowRadius = 5
          intvalue1 = [1, 1, 0.5, 0.3, 0.3, 0]
          intpositionnormalized1 = [0,0.39,0.4,0.6, 0.8, 1]
       )
)
```

Use the `intpositionnormalized1` input field to specify where each point in `intvalue1` occurs on the window, in the normalized length of the window. Therefore, in the above example, from 0 to 0.39*5 away from the window, the value is 1.

An abrupt drop in value from 1 to 0.5 occurs from 0.39*5 to 0.4*5 and so on. The resulting spline function appears as follows:



Rather than using lengths normalized to the window size, absolute lengths in the intensity position field can be specified by using `intposition1` rather than `intpositionnormalized1`.

13.3.8.13 Ray window with spatially varying intensity revisits III

```
RayTrace(...
       CircularWindow(
          windowcenter = (10,12,0)
          WindowDirection = (0,-1,0)
          CellSize = 0.1
          WindowRadius = 5
          intvalue1 = [1, 1, 0.5, 0.3, 0.3, 0]
          intposition1 = [0,1.99, 2, 3, 4, 5]
       )
)
```

The above code achieves a similar result to the code shown previously. `intvalue2` has its own companions, `intposition2` and `intpositionnormalized2`.

In case of a mismatch in length, DESSIS assumes that the vector `intvalueX` only is given and forms a uniform length partition. In case of a nonmonotone increasing position field, either in `intpositionX` or `intpositionnormalizedX`, the non-increasing portion is ignored.

13.4 Optical generation by transfer matrix approach

The DESSIS option OPTIK calculates the propagation of plane waves through layered media by using a transfer matrix approach. An extension for inverted pyramid structures as they are used for high efficiency solar cells allows for the modeling of light propagation in such structures by appropriately transforming them into planar-layered media with similar optical properties.

13.4.1 Physical model

In the underlying model of the optical carrier generation rate, monochromatic plane waves with arbitrary angles of incidence and polarization states penetrating a number of planar, parallel layers are assumed. Each layer must be homogeneous, isotropic, and optically linear. In this case, the amplitudes of forward and backward running waves $A_j^{\pm}$ and $B_j^{\pm}$ in each layer in Figure 15.61 on page 15.267 are calculated with help of transfer matrices.

These matrices are functions of the complex wave impedances Z_j given by $Z_j = n_j \cdot \cos\Theta_j$ in the case of E polarization (TE) and by $Z_j = n_j/(\cos\Theta_j)$ in the case of H polarization (TM). Here, n_j denotes the complex index of refraction and Θ_j is the complex counterpart of the angle of refraction ($n_0 \cdot \sin\Theta_0 = n_j \cdot \sin\Theta_j$).

Real and complex parts of the complex refractive index $\tilde{n} = n + ik$ can be defined using the keywords `Refract` and `Absorption` in the `Layer` section of the input file or can be read off the OptikDB definable in DESSIS parameter file. The transfer matrix of the interface between layers j and $j+1$ is defined by:

$$T_{j,j+1} = \frac{1}{2Z_j} \cdot \begin{bmatrix} Z_j + Z_{j+1} & Z_j - Z_{j+1} \\ Z_j - Z_{j+1} & Z_j + Z_{j+1} \end{bmatrix} \tag{15.299}$$

The propagation of the plane waves through layer j can be described by the transfer matrix:

$$T_j(d_j) = \begin{bmatrix} \exp\left(2\pi i\, n_j \cos\Theta_j \frac{d_j}{\lambda}\right) & 0 \\ 0 & \exp\left(-2\pi i\, n_j \cos\frac{\Theta_j d_j}{\lambda}\right) \end{bmatrix} \tag{15.300}$$

with the thickness d_j of layer j and the wavelength λ of the incident light. The transfer matrices connect the amplitudes of Figure 15.61 on page 15.267 as follows:

$$\begin{pmatrix} B_j^+ \\ A_j^+ \end{pmatrix} = T_{j,j+1} \cdot \begin{pmatrix} A_{j+1}^- \\ B_{j+1}^- \end{pmatrix}$$
$$\begin{pmatrix} A_j^- \\ B_j^- \end{pmatrix} = T_j(d_j) \cdot \begin{pmatrix} B_j^+ \\ A_j^+ \end{pmatrix} \tag{15.301}$$

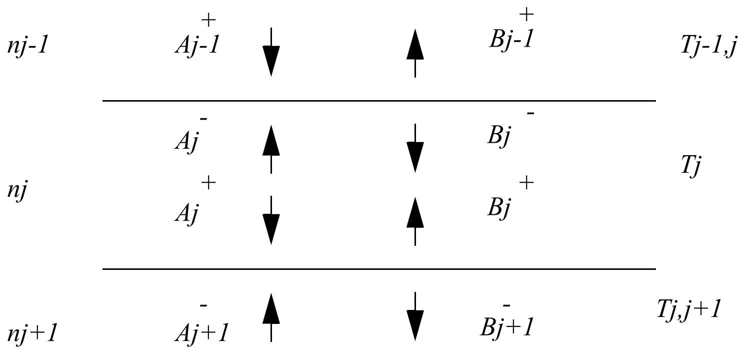


Figure 15.61 Wave amplitudes in a layered medium and transfer matrices connecting them

It is assumed that there is no backward-running wave behind the layered medium, and the intensity of the incident radiation is known. Therefore, the amplitudes $A_j^{\pm}$ and $B_j^{\pm}$ at each interface can be calculated with appropriate products of transfer matrices. For both cases of polarization, the intensity in layer j at a distance d from the upper interface $(j, j+1)$ is given by:

$$I_{T(E,\,TM)}(d) = \frac{\Re(Z_j)}{\Re(Z_0)} \cdot \left\| T_j(d) \cdot \begin{pmatrix} A_j^- \\ B_j^- \end{pmatrix} \right\|^2 \tag{15.302}$$

with the proper wave impedances. If δ is the angle between the vector of the electric field and the plane of incidence, the intensities have to be added according to:

$$I(d) = a\, I_{\mathrm{TE}}(d) + (1-a)\, I_{\mathrm{TE}}(d) \tag{15.303}$$

where $a = \cos^2\delta$.

One of the layers must be the electrical active silicon layer where the optical charge carrier generation rate G^{opt} is calculated. The rate is proportional to the photon flux $\Phi(d) = I(d)/\hbar\omega$.

In the visible and ultraviolet region, the photon energy $\hbar\omega$ is greater than the band gap of silicon. In this region, the absorption of photons by excitation of electrons from the valence to the conduction band is the dominant absorption process for nondegenerate semiconductors. Far from the absorption threshold, the absorption is considered to be independent of the free carrier densities and doping. Therefore, the silicon layer is considered to be a homogeneous region.

The absorption coefficient α is the relative rate of decrease in light intensity along its path of propagation due to absorption. This decrease must be distinguished from variations caused by the superposition of waves. Therefore, the rate of generated electron–hole pairs is:

$$G_0^{\mathrm{opt}} = \alpha\,\eta\,\frac{I(d)}{\hbar\omega} \tag{15.304}$$

where the absorption coefficient α is given by the imaginary part of $4\pi\,Z_{\mathrm{Si}}/\lambda$. The quantum yield η is defined as the number of carrier pairs generated by one photon. Up to photon energies of 3 eV, η equals one. With increasing energy, it increases linearly in first approximation to a value of 3 at 6 eV [5][6]. The additional pairs are created by impact ionization.

The quantum yield is given by:

$$\eta = \begin{cases} 1 + 33.5(0.45\mu\mathrm{m} - \lambda[\mu\mathrm{m}])^2 & \text{for} \quad \lambda < 0.450\mu\mathrm{m} \\ 1 & \text{else} \end{cases} \tag{15.305}$$

unless it is defined using the keyword `QuantumYield` in the `OpticalGeneration` section of the input file.

Under the influence of ultraviolet radiation, the generated electron-hole pairs possess kinetic energies of at least 1 eV, that is, 40 times the thermal energy k_BT. As drift-diffusion equations handle only carriers in thermal equilibrium, the generated electron-hole pairs must not be taken into account in the simulation until they are thermalized. During cooling, they diffuse from the location of generation. Therefore, the generation rate G_0^{opt} must be spread out with a suitable weight function. The spreading is implemented with two weight functions, the stepwise constant function:

$$c(x) = \begin{cases} \dfrac{1}{2\lambda_{\mathrm{sp}}} & -\lambda_{\mathrm{sp}} \le x \le \lambda_{\mathrm{sp}} \\ 0 & \text{otherwise} \end{cases} \tag{15.306}$$

and the Gaussian function:

$$g(x) = \frac{1}{\lambda_{\mathrm{sp}}\sqrt{\pi}} \cdot \exp\left[-\left(\frac{x}{\lambda_{\mathrm{sp}}}\right)^2\right] \tag{15.307}$$

Both functions are normalized:

$$\int_{-\infty}^{+\infty} c(x)\,dx = \int_{-\infty}^{+\infty} g(x)\,dx = 1 \tag{15.308}$$

The modified optical generation G^{opt} is the convolution of G_0^{opt} from (Eq. 15.304) and a weight function, where those carriers that would diffuse out of the silicon layer are mirrored back with a loss factor β :

$$G^{opt}(\mathrm{d}) = \left(\int_0^{\infty} G_0^{\mathrm{opt}}(x)w(\mathrm{d}(-x'))dx'\right) + (1-\beta)\int_0^{\infty} G_0^{\mathrm{opt}}(x)w(\mathrm{d}+x')(dx') \tag{15.309}$$

Here, d is the distance from the upper boundary of the silicon layer and w is one of the weight functions. The characteristic length λ_{sp} can be calculated with a suggested random walk model [7]:

$$\lambda_{\mathrm{sp}} = \sqrt{\frac{2}{3}N_{\mathrm{ph}}}\ \lambda_{\mathrm{ph}} \tag{15.310}$$

where $\lambda_{\mathrm{ph}} = 5.5\ \mathrm{nm}$ is the average mean free path for phonon scattering and N_{ph} is the number of phonons generated during thermalization.

Assuming that optical phonon scattering and impact ionization are the determining mechanisms in the thermalization process, the number of phonons N_{ph} generated during this process is given by:

$$N_{\mathrm{ph}} = \frac{1}{2}\,\frac{\hbar\omega - E_{\mathrm{gap}} - (\eta-1)\langle E_{\mathrm{imp}}\rangle}{\langle E_{\mathrm{ph}}\rangle} \tag{15.311}$$

where E_{gap} is the band gap. $\langle E_{\mathrm{imp}}\rangle = 1.5\ \mathrm{eV}$ and $\langle E_{\mathrm{ph}}\rangle = 0.054\ \mathrm{eV}$ [8] are the average impact ionization and phonon energies, respectively.

The weight functions are defined by assigning the keywords `None`, `Constant`, or `Gaussian` to the keyword `Spreading` in the `OpticalGeneration` section of the input file. The characteristic spreading length is calculated according to (Eq. 15.310) and (Eq. 15.311) unless it is defined using the keyword `Lambda` in the `OpticalGeneration` section. The term β is defined with the keyword `Loss` in the `OpticalGeneration` section.

13.4.2 Syntax and implementation

Opening an `OpticalGeneration` section inside the `Physics` section of DESSIS input file switches to the transfer matrix approach in a DESSIS simulation. The syntax is:

```
OpticalGeneration{
        <OpticalGeneration Options>
        Light(
            <Light Options>
        )
        Layers(
            <Layers Options>
            ( <LayerEntry1> )
            ( <LayerEntry2> )
        )
}
```

All keywords that can be in angle brackets are listed in Table 15.102 to Table 15.105.

Table 15.102 OpticalGeneration options

Keyword	Value	Comments
`QuantumYield`	`float`	
`Lambda`	`float`	Mean free path [µm].
`Loss`	`float`	
`StandingWaves`	`none | include`	
`Area`	`float`	
`Shift`	`vector`	
`MultipleReflections`	`none | include`	

Table 15.103 Light options

Keyword	Value	Comments
`Direction`	`vector`	
`Polarization`	`float in [0,1]`	
`Wavelength`	`list of floats`	For example [0.1 0.2 …] in µm.
`Intensity`	`list of floats`	For example [1e2 1.2e3 …]. The i-th entry of this intensity list is the intensity [W/cm^2] of the light with i-th wavelength in the above wavelength list.

Table 15.104 Layers options

Keyword	Value	Comments
`FacetAngle`	`float`	[degree]
`InitialLayerRefract`	`float`	Refractive index of the layer in which the light wave starts.

Table 15.105 LayerEntry options

Keyword	Value	Comments
`Refract`	`float`	Optional parameters that specify a constant value of refractive index and absorption coefficient that overwrites those values otherwise retrieved from OptikDB.
`Absorption`	`float`	
`Thickness`	`float`	Thickness of current layer [µm].
`Left`	`vector`	'Upper left' corner of the current layer illuminated by light wave. Needed for 1D simulation.
`Right`	`vector`	'Upper right' corner of the current layer illuminated by light wave. Needed for 2D simulation along with `Left` vector.
`Middle`	`vector`	'Upper middle' of current layer illuminated by light wave. Needed for 3D simulation along with `Left` and `Right` vectors.

Table 15.105 LayerEntry options

Keyword	Value	Comments
`TopReflectivity`	`float in [0,1]`	Optional parameters that specify the reflectivity of the current layer that overwrites those values that would have been computed from OptikDB.
`BottomReflectivity`	`float in [0,1]`	

In DESSIS, the results of an optical generation simulation can be incorporated into a device simulation. The syntax is:

```
File {
   ...
   OpticalGenerationFile = <file name>
   ...
}
```

13.5 Optical generation from FDTD simulation (EMLAB)

EMLAB™ is an electromagnetic solver based on the finite difference time domain (FDTD) method. DESSIS can run EMLAB using an interface to generate the tensor grid native to EMLAB from the more general DESSIS grid, to generate an EMLAB input command file, and to load the EMLAB output of an optical generation profile.

This interface is *not* a coupling of Maxwell equations solved by EMLAB and semiconductor equations. Maxwell equations are solve separately, completely outside of DESSIS, and the resulting optical generation is loaded into the continuity equation.

13.5.1 Files of EMLAB generation

Optical generation loading from EMLAB simulation (EMLAB generation) is activated with the command `EMLABGeneration` in the `Physics` section. However, a significant amount of relevant syntax is involved in the `File` section that should be noted beforehand.

To run an EMLAB generation, two files are critical: the EMLAB input command file and the tensor grid corresponding to the device being simulated. Both can be created by using either this DESSIS–EMLAB interface or existing files that are specified by the user in the `File` section.

13.5.1.1 Creating the tensor grid and EMLAB input command file

If no existing files for DESSIS–EMLAB interface are specified in `File` section, the interface creates the requisite files by using the following naming scheme.

In the absence of the command `plot = "XXX"` in the `File` section, the generated tensor grid file is named `EMLAB_grid.ten`. If the command `plot = "XXX"` is present, the tensor grid is called `EMLAB_XXX_grid.ten`, which is the name of the plot file inserted between `EMLAB` and `grid`.

Similarly, the EMLAB input command file is named `EMLAB_input.cmd` or `EMLAB_XXX_input.cmd` depending on the presence or absence of the plot file specification:

```
File {
   Grid = "mytest_mdr.grd"
   Doping = "mytest_mdr.dat"
   Plot = "with_air"
}
```

The DESSIS–EMLAB interface with the above `File` section creates `EMLAB_with_air_grid.ten` and `EMLAB_with_air_input.cmd`.

13.5.1.2 User-defined input or tensor grid

If users want to use their own EMLAB input file or EMLAB tensor grid file, specification of their name is performed in the `File` section. Table 15.106 lists the commands available regarding the DESSIS–EMLAB interface.

Table 15.106 EMLAB-related file options

File commands	Description
`EMLABinput`	EMLAB input command file
`EMLABgrid`	Tensor grid for EMLAB input

If the DESSIS–EMLAB interface ‘sees’ `EMLABinput = "<filename>"` in the `File` section, the interface does not create either an input file of its own or a grid file. If it ‘sees’ `EMLABgrid = "<filename>"`, it creates a grid file of its own, but generates an input command file with which to run EMLAB, for example:

```
File {
   Grid       = "mytest_mdr.grd"
   Doping     = "mytest_mdr.dat"
   Plot       = "with_air"
   EMLABinput = "MyEMLABinput.cmd"
}
```

This code runs EMLAB with `MyEMLABinput.cmd` as the command file. It is assumed that by specifying the input command file the user has also provided the tensor grid file. Another example is:

```
File {
   Grid       = "mytest_mdr.grd"
   Doping     = "mytest_mdr.dat"
   Plot       = "with_air"
   EMLABgrid  = "MyEMLABGrid.ten"
}
```

This code creates an EMLAB input file with the information that is provided in the main `EMLABGeneration` area of the `Physics` section, but does not generate a tensor grid file.

13.5.1.3 Log of EMLAB run

The log of an EMLAB execution is stored in `<EMLABinputfile.cmd>.log`. So, if EMLAB undergoes an abnormal execution, refer to this file for more information.

13.5.2 Syntax of EMLAB generation: EMLAB input file

In `EMLABGeneration`, the input language of EMLAB is replicated in varying degree. Table 15.107 lists the arguments that are recognized by DESSIS `EMLABGeneration`.

Table 15.107 Options of EMLABGeneration

Option	Value
`Boundary`	See further descriptions in the tables that follow.
`Excitation`	
`Material`	
`AutoMatGen`	–
`SmoothingFactor`	`= double`
`GridBoundX`	`(double, double)`
`GridBoundY`	`(double, double)`
`GridBoundZ`	`(double, double)`
`NodePerWavelength`	`= <integer>` Minimum number of nodes that is placed in a wavelength in all directions; default is 10.
`NodePerWavelengthX`	`= <integer>` Similar to `NodePerWavelength` but only for the x direction.
`NodePerWavelengthY`	`= <integer>` Similar to `NodePerWavelength` but only for the y direction.
`NodePerWavelengthZ`	`= <integer>` Similar to `NodePerWavelength` but only for the z direction.

13.5.2.1 Boundary

`Boundary` denotes the boundary condition. EMLAB has options for the following boundary conditions:

- First-order and second-order Mur
- Higdon operator (up to fourth-order)
- Perfectly matched layer (PML)
- Periodic boundary condition

Each of these boundary conditions can be restricted to any subset of the six boundary planes. This allows combinations such as periodicity on one direction and absorbing boundary conditions in the remaining directions. However, in the current version of EMLAB, there is one exception to this general concept of combining different types of boundary conditions: PML boundary conditions cannot be combined with other types of boundary conditions. The default boundary condition chosen by EMLAB is the Higdon condition of second-order. Other choices can be made with the `Boundary` statement in the EMLAB command file. The boundary arguments are listed in Table 15.108 on page 15.274.

Table 15.108 Boundary arguments

Argument	Value	Default
`Sides`	`{XMin XMax...ZMax}` or `{X Y Z}` or `{all}`	`all`
`Type`	`Mur`	`Higdon`
	`PeriodicMur`	
	`Higdon`	
	`PML`	
	`Periodic`	
`Order`	`1..4(Hig)-1..2(Mur)`	`2`

NOTE As nonnumeric values assigned to the arguments inside `Boundary` or another `EMLABGeneration` option are strings that are written exactly in the EMLAB input file that is generated during DESSIS execution, there is no DESSIS check for their validity while parsing DESSIS input. EMLAB registers an error upon its execution if illegal input is used. In addition, as these values are strings, they must be in quotation marks (" ").

An example is:

```
Physics { ...
   EMLABGeneration(
      Boundary(
         Type = "Periodic"
         # notice the quotation marks("") for Type argument is assigned
         # a string value.
         Side = "X"
      )
      Boundary(
         Type = "Higdon"
         Side = "Y"
         Order = 2
      )
      Excitation(
         <options>
      )
   )
}
```

13.5.2.2 Excitation

`Excitation` describes the electromagnetic source. Any number of excitations is allowed. At least one `Excitation` section is necessary for DESSIS to generate a tensor grid. The existing interface syntax describes a plane wave striking the device being simulated. `WaveLength` of the plane wave must be specified and it also determines the minimum grid space when the tensor grid is being generated. The unit of wavelength for the DESSIS–EMLAB interface is meter.

For 2D geometry, the direction of the plane wave propagation is defined by `Theta`, the angle the propagation direction makes with the positive y-axis. The range of `Theta` is from 0 to 180. `Phi` and `Psi` should not be specified for 2D.

For 3D geometry, the direction of the plane wave propagation uses three angles: `Theta`, `Phi`, and `Psi`. `Theta` has different meanings in 2D and 3D than in 2D, where it is the angle the wave propagation direction makes with the positive z-axis. The range of `Theta` is still from 0 to 180. `Phi` is the angle that the wave direction makes with the positive x-axis. `Phi` can be a number between 0 and 360. `Psi` is the polarization angle in degrees of the plane wave measured from the direction given by $k \times z$, where k is the wave vector and z is a vector in the z-direction.

Power in the plane wave can be specified as `WavePower` [W/m^2] or `Amplitude` of the electric field [V/m]. When both are specified, `Amplitude` is ignored.

NOTE `WavePower` is not an available option in EMLAB.

Electric field amplitude A is related to wave power P of the plane wave by:

$$A = \sqrt{2Z_0P} \qquad (15.312)$$

where $Z_0 = \sqrt{\frac{\mu_0}{\varepsilon_0}} = 376.73031\ \Omega$.

NOTE The excitation default in the DESSIS–EMLAB interface relies on `automatic`, which is the excitation region option in EMLAB. Of most importance to the DESSIS–EMLAB user is that the ‘surfaces’ that the plane wave hits must be of uniform material (the material can also be a vacuum).

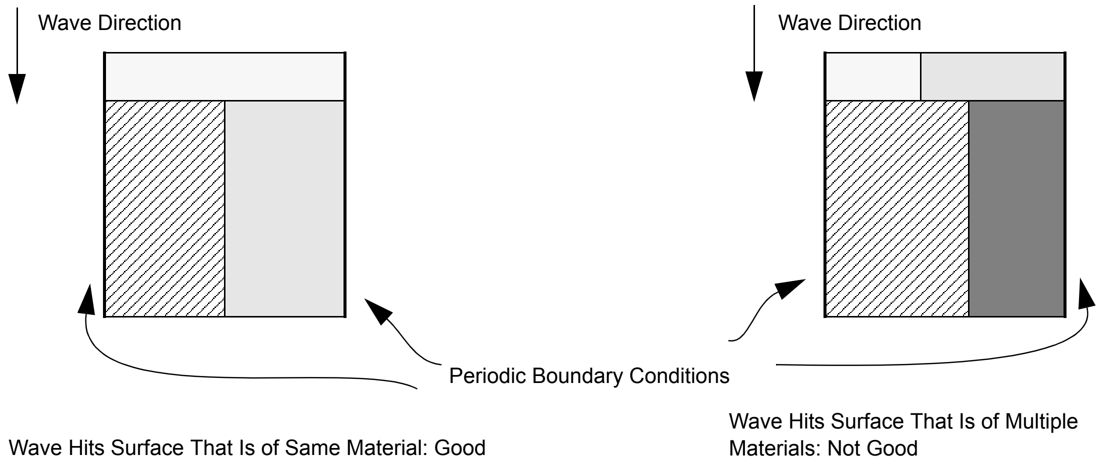

Wave Direction
Wave Direction
Periodic Boundary Conditions
Wave Hits Surface That Is of Same Material: Good
Wave Hits Surface That Is of Multiple Materials: Not Good

Figure 15.62 Device with periodic boundary conditions in y-direction

Figure 15.62 shows a device with periodic boundary conditions in the y-direction. There are two surfaces of interest in this scheme: top x surface and bottom x surface. If the wave is descending, it hits the top x surface, which consists of the same material. For devices without periodic boundary conditions, more than one surface may need to be considered.

Syntax is available that will surround an existing device with a vacuum and, therefore, create a uniform material surface (see Section 13.5.3 on page 15.279).

Table 15.109 lists the options that are used in the `Excitation` statement.

Table 15.109 Excitation arguments

Argument	Value	Default	Unit
`Theta`	`(double)`	–	degree
`Phi`	`(double)` (3D)	–	degree
`Psi`	`(double)` (3D)	–	degree
`Amplitude`	`(double)`	–	V/m
`WavePower`	`(double)`	–	W/m^2
`WaveLength`	`(double)`	–	m

13.5.2.3 Excitation example 1

```
Physics { ...
       EMLABGeneration(
       Boundary(
          Type = "Periodic"
          # notice the quotation marks("") for Type argument is assigned
          # a string value.
          Side = "X"
       )
       Boundary(
          Type = "Higdon"
          Side = "Y"
          Order = 2
       )
       Excitation(
          WavePower = 1e4      # [W/m^2]
          WaveLength = 1e-7    # [m]
          Theta = 180
       )
    )
}
```

This 2D example with its combination of boundary conditions and excitation will create a plane wave ‘descending’ on a device with a power of 1×10^4 W/m^2.

Plane wave excitation is not the only excitation option available in EMLAB. Full EMLAB features are accessed by a user by creating their own EMLAB input file and providing the file name to DESSIS.

13.5.2.4 Material/AutoMatGen

EMLAB has its own database of material properties, `MATDB`. The most common materials used in the semiconductor field are listed. However, if a material is required that is not in `MATDB` or the user does not want to use the values of existing material properties, there is the option to use choice values for the material properties.

Material properties can be specified by either automatically extracting relative permittivity and conductivity from the optical properties of the absorption coefficient and refractive index, or manually entering the material properties in the input file.

Where the main interest is in the optical generation, the material properties relevant to an interface are permittivity, permeability, conductivity, and name. The name given in the `Material` section should match the name in the DESSIS grid file. Table 15.110 lists the arguments for the `Material` section.

Table 15.110 Material arguments

Argument	Description	Default	Unit
`Name`	String	–	–
`Permittivity`	Relative permittivity	1	–
`Permeability`	Relative permeability	1	–
`Conductivity`		0	S/m
`MagneticConductivity`		0	–
`Density`		0	$1/m^3$
`ThermalConductivity`		0	W/m/K
`SpecificHeat`		0	–

An example of overwriting material properties of silicon is:

```
Physics { ...
   EMLABGeneration(
      Boundary(
         Type = "Periodic"
         # notice the quotation marks("") for Type argument is assigned
         # a string value.
         Side = "X"
      )
      Boundary(
         Type = "Higdon"
         Side = "Y"
         Order = 2
      )
      Excitation(
         WavePower = 1e4      # [W/m^2]
         WaveLength = 1e-7    # [m]
         Theta = 180
      )
      Material(
         Name = "Silicon"
         Permittivity = 11.7
         Permeability = 1
         Conductivity = 9671.78
      )
   )
}
```

This example could be used for a more exotic material that `MATDB` does not contain. There is a more convenient way of specifying unknown material properties or overwriting existing ones that are unsatisfactory. The `AutoMatGen` section extracts permittivity and conductivity from internal DESSIS parameters. Permittivity is taken from the DESSIS parameter `Epsilon`. Permeability is assumed to be 1. Conductivity is taken from a specified absorption coefficient model and refractive index models.

Conductivity σ is related to absorption coefficient α, and refractive index n by the following relationship:

$$\sigma = \frac{\alpha n}{Z_0} \tag{15.313}$$

where $Z_0 = \sqrt{\frac{\mu_0}{\varepsilon_0}} = 376.73031\ \Omega$.

Metal, for which absorption and refractive index models do not exist, takes permittivity and permeability to be 1, and conductivity to be 1×10^{30}. The arguments inside `AutoMatGen` are identical to the absorption coefficient and refractive index syntax of `RayTracing`.

NOTE A side effect of the matching between the absorption coefficient and refractive index of `AutoMatGen` and those of `RayTracing` is that the unit of the constant absorption coefficient is cm^{-1}, not m^{-1}.

Table 15.111 AutoMatGen arguments

Argument	Description
`SemAbs =` α or `SemAbs(value =` α `)`	Both statements specify a constant value α for the absorption coefficient. This definition does not require the wave properties of the beam (wavelength or photon energy). Unit is cm^{-1}.
`SemAbs(model = Parameter)`	The absorption coefficient is computed according to the DESSIS parameter file. This is the default.
`SemAbs(model = RSS)`	Use the silicon absorption model [155].
`SemAbs(model = ODB)`	In this case, DESSIS takes the absorption coefficient from the optical database file `optikdata`.
`SemAbs(model = "<PMI Model Name>")`	Use the mentioned PMI absorption coefficient model.
`RefractiveIndex(value =` n `)`	Specifies a constant value n for refractive index.
`RefractiveIndex(model = Parameter)`	The refractive index is computed according to the DESSIS parameter file.
`RefractiveIndex(model = ODB)`	Takes the value of the refractive index from the table-based internal DESSIS optical database `OptikDB`.
`RefractiveIndex(model = "<PMI Model Name>")`	Use the mentioned PMI refractive index model.

The following example demonstrates the usage of `AutoMatGen`:

```
Physics { ...
   EMLABGeneration(
      Boundary(
         Type = "Periodic"
         # notice the quotation marks("") for Type argument is assigned
         # a string value.
         Side = "X"
      )
      Boundary(
         Type = "Higdon"
         Side = "Y"
         Order = 2
      )
      Excitation(
         WavePower = 1e4      # [W/m^2]
```

```
            WaveLength = 1e-7    # [m]
            Theta = 180
         )
         AutoMatGen(
            SemAbsorption(model = parameter)
            RefractiveIndex(model = ODB)
         )
      )
   }
```

This example uses the DESSIS parameter absorption coefficient model and refractive index taken from `OptikDB` to write out a `Material` section in the input command file for EMLAB that looks like:

```
   Material {
      Name = Metal
      Permittivity = 1
      Permeability = 1
      Conductivity = 1e+30
      MagneticConductivity = 0
      Density = 0
      ThermalConductivity = 0
      SpecificHeat = 0
   }
   Material {
      Name = Si3N4
      Permittivity = 7.5
      Permeability = 1
      Conductivity = 7681.36
      MagneticConductivity = 0
      Density = 0
      ThermalConductivity = 0
      SpecificHeat = 0
   }
   Material {
      Name = Silicon
      Permittivity = 11.7
      Permeability = 1
      Conductivity = 9671.78
      MagneticConductivity = 0
      Density = 0
      ThermalConductivity = 0
      SpecificHeat = 0
   }
```

13.5.3 EMLAB generation: Tensor grid, syntax, and algorithm

Before describing the syntax relating to tensor grid generation, a brief description of the algorithm is required. The x-, y-, and z-coordinates of vertices of the original DESSIS mesh are queued separately for consideration.

First, the x-coordinates of the vertices are considered. As each x-coordinate of a vertex is popped off the queue of all x-coordinates, some x-coordinates are placed in a second queue. If the location of the x-coordinate is further than `d_min` from other coordinates already in the second queue, it is put in the second queue or it is discarded.

Any gaps between the members of this sparser second queue is filled by inserting more x-coordinates, so that no member of this second queue is further than `d_max` from its neighbors.

The maximum grid space, `d_max`, in the resulting tensor mesh is determined by the wavelength specified in the `Excitation` section: `d_max` is `wavelength`/20 and `d_min` is `d_max`/2. `GridBound*` arguments can limit the range of coordinates, and the `smoothingfactor` argument can make the distribution of coordinates more uniform. Their functionality and syntax are discussed in the following sections.

The coordinates in this second queue note where the tensor grid's vertices if x direction will be. The same algorithm is performed for the y-coordinates and z-coordinates.

13.5.3.1 GridBoundX, GridBoundY, and GridBoundZ

Arguments relating to the tensor grid that DESSIS generates for EMLAB are `GridBound*` series arguments that specify the domain that is represented by the tensor grid that is generated and, therefore, the physical domain where the EMLAB simulation will run. `GridBound*` is followed by a two-entry vector in parentheses, for example, `GridBoundX(1, 5)` binds the x-grid in the interval [1, 5].

Figure 15.63 demonstrates this concept. Similarly, `GridBoundZ` 'bind' the resulting EMLAB device in the z-direction. If `GridBoundY` is not specified, as in the second figure, the entire DESSIS device in the y-direction will be present in the EMLAB device and nothing else. The region that does not correspond to any part of the original device is considered to be a vacuum. The `GridBound*` series of arguments has two functions. First, by binding the simulation area to a smaller region where all the interesting events will occur, the DESSIS–EMLAB interface can speed up the EMLAB run. Second, for the excitation option used by the DESSIS–EMLAB interface, the very 'top' surface where the plane wave will first come in contact with the device needs to be uniform in material. By 'padding' the top with vacuum for some uneven-surfaced devices, such as a silicon dioxide lens, there can be a layer of uniform material covering the top surface without the user having to alter the original device. Further, it may be desirable to have a vacuum layer where the excitation plane wave of devices, such as solar cells and photo detectors, will probably come from.

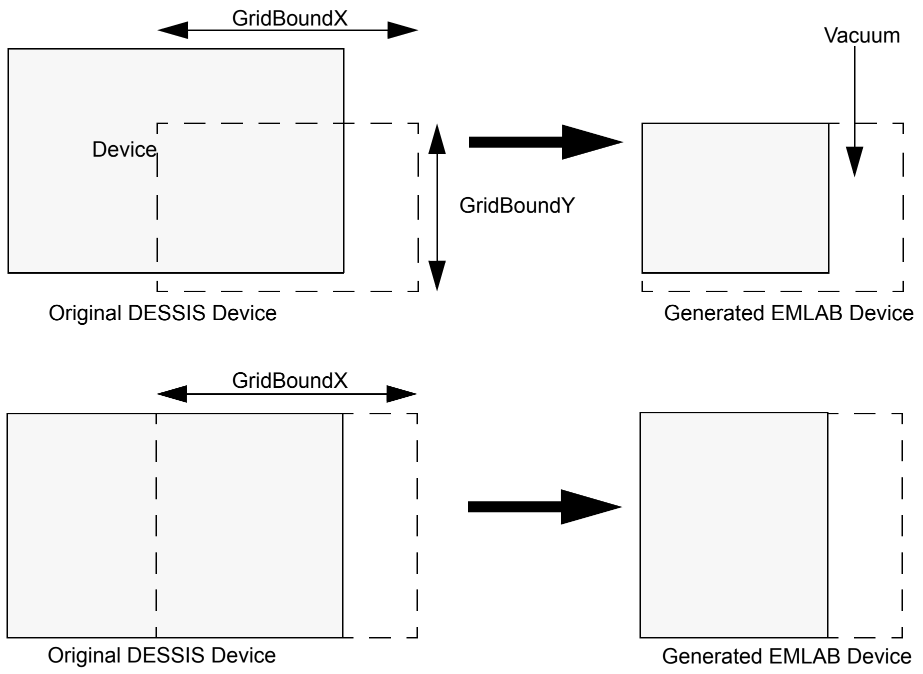


Figure 15.63 Illustration of GridBoundX, GridBoundY, and GridBoundZ

An example of code is:

```
Physics { ...
   EMLABGeneration(
      Boundary(
         Type = "Periodic"
         # notice the quotation marks("") for Type argument is assigned
         # a string value.
         Side = "X"
      )
      Boundary(
         Type = "Higdon"
         Side = "Y"
         Order = 2
      )
      Excitation(
         WavePower = 1e4      # [W/m^2]
         WaveLength = 1e-7    # [m]
         Theta = 180
      )
      AutoMatGen(
         SemAbsorption(model = parameter)
         RefractiveIndex(model = ODB)
      )
      SmoothingFactor = 1.5
      GridBoundX(0, 10)
      GridBoundY(-1, 12)
   )
}
```

13.5.3.2 SmoothingFactor

The argument `smoothingfactor`, which relates to the tensor grid that DESSIS generates for EMLAB, regulates the grid spacing in the generated tensor mesh.

Closer to uniform, a mesh with grid spacing that is equal, a tensor mesh is, more stable the FDTD simulation is. Although complete uniformity is not necessary, a large difference between grid spacing in adjacent nodes is not good. The argument `smoothingfactor = s` in the `EMLABGeneration` statement forces the ratio of adjacent grid spacings in the generated mesh within the smoothing factor `s`, which is a number strictly greater than 1.1. If the smoothing factor is less than 1.1, DESSIS issues a warning and uses 1.1. The following is a sample code containing the `Smoothing` option of `EMLABGeneration`:

```
Physics { ...
   EMLABGeneration(
      Boundary(
         Type = "Periodic"
         # notice the quotation marks("") for Type argument is assigned
         # a string value.
         Side = "X"
      )
      Boundary(
         Type = "Higdon"
         Side = "Y"
         Order = 2
      )
      Excitation(
         WavePower = 1e4      # [W/m^2]
         WaveLength = 1e-7    # [m]
         Theta = 180
      )
```

```
         AutoMatGen(
            SemAbsorption(model = parameter)
            RefractiveIndex(model = ODB)
         )
         Smoothingfactor = 1.5
      )
   }
```

13.6 Optical AC analysis

An optical AC analysis calculates the quantum efficiency as a function of the frequency of the optical signal. The method is based on the AC analysis technique and provides real and imaginary parts of the quantum efficiency versus the frequency.

During an optical AC analysis, a small perturbation of the incident wave power δP_0 is applied. Therefore, the photo generation rate is perturbated as $G^{opt} + \delta G^{opt} e^{i\omega t}$, where $\omega = 2\pi f$ (f is the frequency) and δG^{opt} is an amplitude of a local perturbation. The resulting small-signal device current perturbation δI_{dev} is the sum of real and imaginary parts, and the expressions for the quantum efficiency are:

$$\begin{aligned} \eta &= \frac{Re[\delta I_{dev}]/q}{\delta P^{tot}\lambda/hc} \\ C_{opt} &= \frac{1}{\omega} \cdot \frac{Im[\delta I_{dev}]/q}{\delta P^{tot}\lambda/hc} \\ \delta P^{tot} &= \int_S \delta P_0 ds \end{aligned} \tag{15.314}$$

where the quantity $\delta P^{tot}\lambda/hc$ gives a perturbation of the total number of photons and $Re[\delta I_{dev}]/q$ is a perturbation of the total number of electrons at an electrode. As a result, for each electrode, DESSIS places two values into the AC output file, `photo_a` and `photo_c`, that correspond to η and C_{opt}, respectively. To start the optical AC analysis, add the keyword `Optical` in the `ACCoupled` statement, for example:

```
ACCoupled ( StartFrequency=1.e4 EndFrequency=1.e9
   NumberOfPoints=31 Decade  Node(a c) Optical )
   { poisson electron hole }
```

NOTE If an element is excluded (`Exclude` statement) in optical AC (this is usually the case for voltage sources in regular AC simulation), it means that this element is not present in the simulated circuit and, correspondingly, it provides zero AC current for all branches that are connected to the element. Therefore, do *not* exclude voltage sources.

CHAPTER 14 Single event upset (SEU)

14.1 Alpha particles

See Section 14.3 on page 15.288.

14.1.1 Syntax and implementation

Specify the keyword `AlphaParticle` in the `Physics` section of the DESSIS input file:

```
Physics { ...
        AlphaParticle (<optional keywords>)
}
Plot { ...
        AlphaCharge
}
```

Table 15.112 lists the keyword options for alpha particles.

Table 15.112 Keyword options for AlphaParticle command

Keyword	Description
`Energy = <float>`	Defines the energy of the alpha particle.
`Time = <float>`	Defines the time at which the charge generation peaks in the device.
`Location = <x,y,z>`	Defines the point where the alpha particle enters the device [μm].
`Direction = <vector>`	Vector (x, y, z) defines the direction of motion of the particle.

Table 15.113 Coefficients for carrier generation by alpha particles

Symbol	Parameter name	Default value	Unit
s	`s`	2×10^{-12}	s
w_t	`wt`	1×10^{-5}	cm
c_2	`c2`	1.4	1
a	`alpha`	90	cm^{-1}
α_2 α_3	`alpha2` `alpha3`	5.5×10^{-4} 2×10^{-4}	cm cm
E_P	`Ep`	3.6	eV

a_0
a_1
a_2
	`a0`
	`a1`
	`a2`
		-1.033×10^{-4}
		2.7×10^{-10}
		4.33×10^{-17}
			cm
			cm/eV
			cm/eV^2

14.1.1.1 Model description

The generation rate caused by an alpha particle with energy E is computed by:

$$G(u, v, w, t) = \frac{a}{\sqrt{2\pi} \cdot s} e^{-\frac{1}{2}\left(\frac{t-t_m}{s}\right)^2} e^{-\frac{1}{2}\left(\frac{v^2+w^2}{w_t^2}\right)} \times \left[c_1 e^{\alpha u} + c_2 e^{-\frac{1}{2}\left(\frac{u-\alpha_1}{\alpha_2}\right)^2}\right] \quad (15.315)$$

if $u < \alpha_1 + \alpha_3$, and by:

$$G(u, v, w, t) = 0 \quad (15.316)$$

if $u \geq \alpha_1 + \alpha_3$ [89]. In this case, u is the coordinate along the particle path and v and w are coordinates orthogonal to u. The direction and place of incidence are defined in the `Physics` section of the input file with the keywords `Direction` and `Location`, respectively. Parameter t_m is the time of the generation peak defined by the keyword `Time`. A Gaussian time dependence can also be used to simulate the typical generation due to pulsed laser or electron beams.

The maximum of the Bragg peak, α_1, is fitted to data [90] by a polynomial function:

$$\alpha_1 = a_0 + a_1 E + a_2 E^2 \quad (15.317)$$

The parameter c_1 is given by:

$$c_1 = e^{\alpha(\alpha_1[10\text{MeV}] - \alpha_1[E])} \quad (15.318)$$

The scaling factor a is determined from:

$$\int_0^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} G(u, v, w, t)\,dt\,dw\,dv\,du = \frac{E}{E_p} \quad (15.319)$$

where E_p is the average energy needed to create an electron–hole pair. The remaining parameters are listed in Table 15.112 on page 15.283. They are available in the parameter file `dessis.par` and are valid for alpha particles with energies between 1 MeV and 10 MeV. The generation by alpha particles cannot be used except in transient simulations. The amount of electron–hole pairs generated before the initial time of the transient is added to the carrier densities at the beginning of the simulation. The charge density $\int_{-\infty}^{\infty} dtG$ is plotted by using the keyword `AlphaCharge` in the `Plot` statement of the input file.

An option to improve the spatial integration of the charge generation is presented in Section 14.3 on page 15.288.

14.2 Heavy ions

The single event upset (SEU) (or bit flip or soft error) is a switch of the accidental, occasional, localized, logical state of a device. The SEU can be observed in bipolar and MOS technologies. This phenomenon is important in satellites, for example, there are 180 upsets/year on GALILEO.

When a heavy ion penetrates into a device structure, it loses energy and creates a trail of electron–hole pairs.

These additional electrons and holes may cause a large enough current to switch the logic state of a device, for example, that of a memory cell. Important factors are:

- The energy and type of the ion.
- The angle of penetration of the ion.
- The relation between the lost energy or linear energy transfer (LET) and the number of pairs created.

14.2.1 Syntax and implementation

The simulation of an SEU caused by a heavy ion impact is activated by using the keyword `HeavyIon` in an appropriate `Physics` section:

```
Physics {
   HeavyIon (<keyword_options>) }
```

Table 15.114 describes the keyword options for `HeavyIon`. The generation rate by the heavy ion is generally used in transient simulations. The number of electron–hole pairs generated before the initial time of the transient is added to the carrier densities at the beginning of the simulation. The total charge density is plotted using the keyword `HeavyIonCharge` in the `Plot` statement of the input file.

Table 15.114 Keyword options for HeavyIon command

Keyword	**Description**
`Time = <float>`	Defines the time [s] at which the ion penetrates the device.
`Location = <x,y,z>`	Defines the point [µm] where the heavy ion enters the device.
`Direction = <vector>`	Vector (x,y,z) defines the direction of motion of the ion.
`Gaussian`	Choose a Gaussian shape for the spatial distribution, R(w).
`Exponential`	Choose an exponential shape for R(w) (true by default).
`LET_f = [float1, float2, ...]`	Defines the linear energy transfer (LET) function of the heavy ion (in pairs/cm^3 by default or pC/µm if the `PicoCoulomb` option is selected). `LET_f = [float1]` is the `LET` value for `Length = [float1]`.
`Wt_hi = [float1, float2, ...]`	Defines the characteristic distance, $w_t(l)$ (in cm by default or µm if the `PicoCoulomb` option is toggled). `Wt_hi = [float1]` is the distance value corresponding to `Length = [float1]`.
`Length = [float1, float2, ...]`	Defines the length l, where `Wt_hi` and `Let_f` will be specified (`Length` in cm by default or µm if the `PicoCoulomb` option is selected).
`PicoCoulomb`	Enforces the picocoulomb/micron unit in `LET_f`. The default unit for `LET_f` is pairs/cm^3.

NOTE If the value of `Wt_hi` is 0, then uniform generation is selected. If the value of `LET_f` is 0, the keyword `LET_f` can be ignored.

14.2.2 Model description

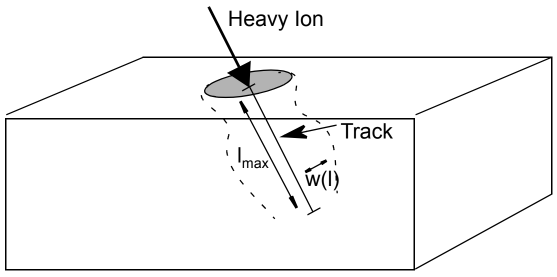


Figure 15.64 A heavy ion penetrating into semiconductor; its track is defined by a length and the transverse spatial influence is assumed to be symmetric about the track axis

A simple model for the heavy ion impinging process is shown in Figure 15.64. The generation rate caused by the heavy ion is computed by:

$$G(l, w, t) = G_{LET}(l) \times R(w, l) \times T(t) \tag{15.320}$$

if $l < l_{\max}$ ($l_{\max}$ is the length of the track), and by:

$$G(l, w, t) = 0 \tag{15.321}$$

if $l \geq l_{max}$. $R(w)$ and $T(t)$ are functions describing the spatial and temporal variations of the generation rate. $G_{LET}(l)$ is the linear energy transfer generation density and its unit is pairs/cm^3.

$T(t)$ is defined as a Gaussian function:

$$T(t) = \frac{2 \cdot \exp\left(-\left(\frac{t - time}{s_{hi}}\right)^2\right)}{s_{hi}\sqrt{\pi}\left(1 - erf\left(\frac{time}{s_{hi}}\right)\right)} \tag{15.322}$$

where *time* is the moment of the heavy ion penetration (see the keyword `Time` in Table 15.114 on page 15.285), and s_{hi} is the characteristic value of the Gaussian (see `s_hi` in Table 15.116 on page 15.287).

The spatial distribution, $R(w, l)$, can be defined as an exponential function (default):

$$R(w, l) = e^{-\left(\frac{w}{w_t(l)}\right)} \tag{15.323}$$

or a Gaussian function:

$$R(w, l) = e^{-\left(\frac{w}{w_t(l)}\right)^2} \tag{15.324}$$

where w is a radius defined as the perpendicular distance from the track. The characteristic distance w_t is defined as `Wt_hi` in the `HeavyIon` statement and can be a function of the length l (see Table 15.114).

The linear energy transfer (LET) generation density, $G_{LET}(l)$, is given by:

$$G_{LET}(l) = a_1 + a_2 \times l + a_3 e^{a_4 \times l} + k'\left[c_1 \times (c_2 + c_3 * l)^{c_4} + LET\_f(l)\right] \tag{15.325}$$

where $LET\_f(l)$ (defined likewise by the keyword `LET_f`) is a function of the length l. Example 2 in Section 14.2.3 on page 15.288 shows how one can use an array of values to specify the length dependence of $LET\_f(l)$. A linear interpolation is used for values between the array entries of `LET_f`.

There are two options for the units of `LET_f`: pairs/cm^3 (default) or pC/µm (activated by the keyword `PicoCoulomb`). Depending on the units of `LET_f` chosen, k' takes on different values in order to make the above equation dimensionally consistent. The appropriate values of k' for different device dimensions are summarized in Table 15.115.

Table 15.115 Setting correct k' to make LET generation density equation dimensionally consistent

Condition	Two-dimensional device	Three-dimensional device
`LET_f` has units of pairs/cm^3 for $R(w,l)$ is exponential or Gaussian	$k' = k$	$k' = k$
`LET_f` has units of pC/µm and $R(w,l)$ is exponential	$k' = \frac{k}{2w_t d}$ $d = 1\,\mu\text{m}$	$k' = \frac{k}{2\pi w_t^2}$
`LET_f` has units of pC/µm and $R(w,l)$ is Gaussian	$k' = \frac{k}{\sqrt{\pi} \times w_t \times d}$ $d = 1\,\mu\text{m}$	$k' = \frac{k}{\pi w_t^2}$

Great care must also be exercised to choose the correct units for `Wt_hi` and `Length`. The default unit of `Wt_hi` and `Length` is centimeter. If the keyword `PicoCoulomb` is specified, the unit becomes µm. Examples illustrating the correct unit use are shown in Section 14.2.3. The other coefficients used in (Eq. 15.325) are listed in Table 15.116 with their default values, and they can be adjusted in the DESSIS parameter file.

Table 15.116 Coefficients for carrier generation by heavy ion (in DESSIS parameter file)

	s_{hi}	a_1	a_2	a_3	a_4	k	c_1	c_2	c_3	c_4
Keyword	`s_hi`	`a_1`	`a_2`	`a_3`	`a_4`	`k_hi`	`c_1`	`c_2`	`c_3`	`c_4`
Default value	2e-12	0	0	0	0	1	0	1	0	1
Default unit	s	pairs/cm^3	pairs/cm^3/cm	pairs/cm^3	cm^{-1}	1	pairs/cm^3	1	cm^{-1}	1
Unit if `Picocoulomb` **is chosen**	s	pairs/cm^3	pairs/cm^3/µm	pairs/cm^3	µm^{-1}	1	pC/µm	1	µm^{-1}	1

14.2.3 Examples: Heavy ions

14.2.3.1 Example 1

The track has a constant LET_f value of 0.2 pC/µm across the track. The track length is 1 µm (l_{max} = 1 µm) and the heavy ion crosses the device at the time 0.1 pS. The unit of LET_f is pC/µm and the spatial distribution is Gaussian. Since PicoCoulomb was chosen, the values of Length and wt_hi are expressed in terms of µm. The keyword HeavyIonChargeDensity in the Plot statement allows users to plot the charge density generated by the ion.

```
Physics { Recombination ( SRH(DopingDep) )
   Mobility (DopingDep Enormal HighFieldSaturation)
   HeavyIon (
      Direction=(0,1)
      Location=(1.5,0)
      Time=1.0e-13
      Length=1
      wt_hi=3
      LET_f=0.2
      Gaussian
      PicoCoulomb )
}
Plot { eDensity hDensity ElectricField HeavyIonChargeDensity
}
```

14.2.3.2 Example 2

The LET_f and radius (wt_hi) values are functions of the position along the track (in this case, l_{max} = 1.7 µm = 1.7×10^{-4} cm). Values in between the array entries are linearly interpolated. The unit of LET_f is pairs/cm^3 (because the keyword Picocoulomb is not used), and the unit for Length and wt_hi is centimeter. For each value of length, there is a corresponding value of LET_f and a value for the radius. The spatial distribution in the perpendicular direction from the track is exponential.

```
Physics { Recombination ( SRH(DopingDep) )
   HeavyIon (
      Direction=(0,1)
      Location=(1.5,0)
      Time=1.0e-13
      Length = [1e-4 1.5e-4 1.6e-4 1.7e-4]
      LET_f = [1e6 2e6 3e6 4e6]
      wt_hi = [0.3e-4 0.2e-4 0.25e-4 0.1e-4]
      Exponential )
}
```

14.3 Improved alpha particle/heavy ion generation rate integration

Accurate integration of alpha particle or heavy ion generation rates is very important for predictive modeling of SEU phenomena. By default, in DESSIS, the integration of the generation rate over the control volume associated with each vertex in the mesh is performed under the assumption that the generation rate is constant inside the vertex control volume and equal to the generation rate value at the vertex. As alpha particle or heavy ion generation rates can change very rapidly in space, the approximation error with such an approach may lead to large errors on a coarse mesh (in particular, the method does not guarantee charge conservation).

To eliminate this source of numeric error, an improved spatial integration can be performed. The procedure used in DESSIS for optical generation (see Chapter 13 on page 15.243) extends to the alpha particle/heavy ion case. Each control volume is covered by a set of small rectangular boxes and the generation rate is integrated numerically inside these boxes. To activate this procedure, the keyword `RecBoxIntegr` must be specified in the `Math` section. The same keyword is used as for optical generation, with the same set of default parameters. By changing the default parameters, the user can also control the accuracy of the integration (see Section 13.1 on page 15.243).

CHAPTER 15 Noise and fluctuation analysis

15.1 Overview

Noise analysis is concerned with deviations from the average behavior that occur dynamically in an average device. Fluctuation analysis is concerned with (static) deviations of device properties from an average device. Section 15.2 explains how to perform noise and fluctuation analysis. Deviations occur for a multitude of reasons; for each of them, a physical model is required, and the availability of models determines the kind of noise or fluctuation that can be modeled. Section 15.3 on page 15.293 discusses the models DESSIS offers. Section 15.4 on page 15.295 discusses the background of the impedance field method [119], and Section 15.5 on page 15.296 summarizes the data available for visualization.

15.2 Performing noise and fluctuation analysis

DESSIS models noise and fluctuations of device properties in a unified manner as an extension of small-signal analysis (see Section 3.8.3 on page 15.117). DESSIS computes the variances and correlation coefficients for the voltages at selected circuit nodes, assuming the net current to these nodes is fixed. As the computation is performed in frequency space, the computed quantities are called the noise voltage spectral densities.

To use noise and fluctuation analysis, first specify the physical models for the microscopic origin of the deviations (called the local noise sources, LNS) as options to the keyword `Noise` in the `Physics` section of the DESSIS command file:

```
Physics {
   ...
   Noise ( <Noisemodels> )
}
```

where `<Noisemodels>` is a list that specifies any number of noise models listed in Table 15.117. As for other physical models, noise sources can be specified regionwise or materialwise.

Table 15.117 Noise sources

Noise model	Keyword	Options

Diffusion noise	`DiffusionNoise`	`LatticeTemperature`
		`eTemperature`
		`hTemperature`
		`e_h_Temperature`

Generation–recombination (GR) noise	`MonopolarGRNoise`	–
	`FlickerGRNoise`	–
Random dopant fluctuations	`Doping`	–

Second, use the `ObservationNode` option to the `ACCoupled` statement to specify the device nodes for which the noise voltage spectral densities are desired, for example:

```
ACCoupled (
   StartFrequency = 1.e8 EndFrequency = 1.e11
   NumberOfPoints = 7 Decade
   Node (n_source n_drain n_gate)
   Exclude (v_drain v_gate)
   ObservationNode (n_drain n_gate)
   ACExtraction = "mos"
   NoisePlot = "mos"
   ){
   poisson electron hole contact circuit
   }
}
```

The keyword `ObservationNode` enables noise analysis (in this case, for the nodes `n_drain` and `n_gate`). `NoisePlot` specifies a file name prefix for device-specific plots (see Section 15.5 on page 15.296). For more information on the `ACCoupled` statement, see Section 3.8.3 on page 15.117.

NOTE The observation nodes must be a subset of the nodes specified in `Node(...)`.

The results of the analysis are the noise voltage autocorrelation and cross-correlation spectral densities. They appear in the `ACExtraction` file (see Table 15.118).

Table 15.118 Noise voltage spectral densities

Name	Description
`S_V`	Noise voltage spectral density (NVSD)
`S_V_ee` `S_V_hh`	Electron NVSD Hole NVSD
`S_V_eeDiff` `S_V_hhDiff`	Electron NVSD due to diffusion LNS Hole NVSD due to diffusion LNS
`S_V_eeMonoGR` `S_V_hhMonoGR`	Electron NVSD due to monopolar GR LNS Hole NVSD due to monopolar GR LNS
`S_V_eeFlickerGR` `S_V_hhFlickerGR`	Electron NVSD due to flicker GR LNS Hole NVSD due to flicker GR LNS
`S_V_Doping`	NVSD due to random dopant fluctuations
`ReS_VXV` `ImS_VXV`	Real/imaginary parts of the cross-noise voltage spectral density (NVXVSD)

`ReS_VXV_ee`
`ImS_VXV_ee`
`ReS_VXV_hh`
`ImS_VXV_hh`
	Real/imaginary parts of the electron/hole NVXVSD

`ReS_VXV_eeDiff`
`ImS_VXV_eeDiff`
`ReS_VXV_hhDiff`
`ImS_VXV_hhDiff`
	Real/imaginary parts of the electron/hole NVXVSD due to diffusion LNS

Table 15.118 Noise voltage spectral densities

Name	Description

`ReS_VXV_eeMonoGR`
`ImS_VXV_eeMonoGR`
`ReS_VXV_hhMonoGR`
`ImS_VXV_hhMonoGR`
	Real/imaginary parts of the electron/hole NVXVSD due to monopolar GR LNS

`ReS_VXV_eeFlickerGR`
`ImS_VXV_eeFlickerGR`
`ReS_VXV_hhFlickerGR`
`ImS_VXV_hhFlickerGR`
	Real/imaginary parts of the electron/hole NVXVSD due to flicker GR LNS

`ReS_VXV_Doping` `ImS_VXV_Doping`	Real/imaginary parts of the NVXVSD due to random dopant fluctuations

15.3 Noise sources

15.3.1 Diffusion noise

The diffusion noise source (keyword `DiffusionNoise`) available in DESSIS reads:

$$K_{n,n}^{\mathrm{Diff}}(\vec{r}_1, \vec{r}_2, \omega) = 4qn(\vec{r}_1)k_{\mathrm{B}}T(\vec{r}_1)\mu_n(\vec{r}_1)\delta(\vec{r}_1 - \vec{r}_2) \tag{15.326}$$

where n is the electron density and μ_n is the electron mobility. A corresponding expression is used for holes. $K_{n,n}^{\mathrm{Diff}}$ is a diagonal tensor.

T is either the lattice or carrier temperature, depending on the specification in the command file:

```
DiffusionNoise ( <temp_option> )
```

where `<temp_option>` is `LatticeTemperature`, `eTemperature`, `hTemperature`, or `e_h_Temperature`. The default is `LatticeTemperature`. For example, if the following command is specified:

```
Physics {
   Noise ( DiffusionNoise ( eTemperature ) )
}
```

DESSIS uses the electron temperature for the electron noise source and the lattice temperature for the hole noise source. The keyword `e_h_Temperature` forces the corresponding carrier temperature to be used for the diffusion noise source for each carrier type.

15.3.2 Equivalent monopolar generation–recombination noise

An equivalent monopolar generation–recombination (GR) noise source model (keyword `MonopolarGRNoise`) for a two-level, GR process can be expressed as a tensor [120]:

$$K_{n,n}^{\mathrm{GR}}(\vec{r}_1, \vec{r}_2, \omega) = \frac{\vec{J}_n(\vec{r}_1) \otimes \vec{J}_n(\vec{r}_1)}{n} \cdot \frac{4\alpha\tau_{\mathrm{eq}}}{1 + \omega^2 \cdot \tau_{\mathrm{eq}}^2}\delta(\vec{r}_1 - \vec{r}_2) \tag{15.327}$$

where $\underset{\sim}{J}_n$ is the electron current density; n, the electron density; α, a fitting parameter; and τ_{eq}, an equivalent GR lifetime. The expression $\underset{\sim}{J}_n \otimes \underset{\sim}{J}_n$ denotes the outer (dyadic) product. The parameters τ_{eq} and α can be modified in the DESSIS parameter file. A similar expression is used for holes.

15.3.3 Bulk flicker noise

The flicker generation–recombination (GR) noise model (keyword `FlickerGRNoise`) for electrons (similar for holes) is:

$$K_{n,n}^{\mathrm{fGR}}(\vec{r}_1, \vec{r}_2, \omega) = \frac{\underset{\sim}{J}_n(\vec{r}_1) \otimes \underset{\sim}{J}_n(\vec{r}_1)}{n(\vec{r}_1)} \cdot \frac{2\alpha_{\mathrm{H}}}{\pi f \log(\tau_1/\tau_0)} \cdot [\mathrm{atan}(\omega\tau_1) - \mathrm{atan}(\omega\tau_0)]\delta(\vec{r}_1 - \vec{r}_2) \qquad (15.328)$$

where $\underset{\sim}{J}_n$ is the electron current density; n, the electron density; α_{H}, a fit parameter; $\omega = 2\pi f$, the angular frequency; and the time constants fulfill $\tau_0 < \tau_1$. The parameters α_{H}, τ_0, and τ_1 for electrons and holes are accessible in the DESSIS parameter file. With increasing frequency, the noise source changes from constant to $1/f$ behavior close to the frequency $f_1 = 1/\tau_1$ and, ultimately, to $1/f^2$ behavior at $f_0 = 1/\tau_0$.

15.3.4 Random dopant fluctuations

The noise sources for random dopant fluctuations are activated by the `Doping` keyword. The noise source for acceptor fluctuations reads:

$$K_{\mathrm{A}}^{\mathrm{RDF}}(\vec{r}_1, \vec{r}_2, \omega) = N_{\mathrm{A}}(\vec{r}_1)\frac{\Theta(0.5\,\mathrm{Hz} - |f|)}{1\,\mathrm{Hz}}\delta(\vec{r}_1 - \vec{r}_2) \qquad (15.329)$$

Here, $f = \omega/2\pi$ is the frequency and Θ is the unit step function. An analogous expression holds for the noise source for donors, $K_{\mathrm{D}}^{\mathrm{RDF}}$. Physically, the noise sources are static. However, to avoid a δ-function in frequency space, DESSIS spreads the spectral density of the noise source over a 1 Hz frequency interval. (Eq. 15.329) is based on the assumption that individual dopant atoms are completely uncorrelated.

By default, DESSIS neglects the impact of the random dopant fluctuations on mobility and band-gap narrowing. To take their impact into account, specify the `Mobility` and `BandGapNarrowing` options to the `Doping` keyword. For example:

```
Physics { Noise( Doping(Mobility) ) }
```

activates the random dopant fluctuation noise source, taking into account the impact of the fluctuations on mobility.

15.3.5 Noise from SPICE circuit elements

To take into account the noise generated by SPICE circuit elements (see Compact Models, Chapter 1 on page 16.1), specify the `CircuitNoise` option to `ACCoupled`. The form of the noise source for a particular circuit element is defined by the respective compact model.

Due to a restriction of the SPICE noise models, SPICE circuit elements contribute only to the autocorrelation noise. For cross-correlation noise, DESSIS considers SPICE circuit elements as noiseless. Non–SPICE compact circuit elements do not implement noise at all and, therefore, DESSIS always treats them as noiseless.

15.4 Impedance field method

The impedance field method splits noise and fluctuation analysis into two tasks. The first task is to provide models for the noise sources, that is, for the local microscopic fluctuations inside the devices (see Section 15.3 on page 15.293). The selection of the appropriate models depends on the problem. Users have to pick the models according to the kind of noise they are interested in. The second task is to determine the impact of the local fluctuations on the terminal characteristics. To solve this task, the response of the contact voltage to local fluctuation is assumed to be linear. For each contact, Green functions are computed that describe this linear relationship. In contrast to the first task, the second task is purely numeric, as the Green functions are completely specified by the transport model.

A Green function describes the response $G_{\xi}(x, x', \omega)$ of the potential at location x due to a perturbation at location x' with angular frequency ω in the right-hand side of the partial differential equation for solution variable ξ. ξ can be Ψ (see (Eq. 15.19)), n or p (see (Eq. 15.20)), T_n (see (Eq. 15.28)), or T_p (see (Eq. 15.29)).

The implemented models result in the following expression for the noise voltage spectral density:

$$S(x, x', \omega) = \int \underline{G}_n(x, x'', \omega) K_{n,n}^{\mathrm{Diff}}(x'', \omega) \underline{G}_n^*(x', x'', \omega) dx'' \tag{15.330}$$

$$+ \int \underline{G}_n(x, x'', \omega) K_{n,n}^{\mathrm{GR}}(x'', \omega) \underline{G}_n^*(x', x'', \omega) dx'' \tag{15.331}$$

$$+ \int \underline{G}_n(x, x'', \omega) K_{n,n}^{\mathrm{fGR}}(x'', \omega) \underline{G}_n^*(x', x'', \omega) dx'' \tag{15.332}$$

$$+ \int \underline{G}_p(x, x'', \omega) K_{p,p}^{\mathrm{Diff}}(x'', \omega) \underline{G}_p^*(x', x'', \omega) dx'' \tag{15.333}$$

$$+ \int \underline{G}_p(x, x'', \omega) K_{p,p}^{\mathrm{GR}}(x'', \omega) \underline{G}_p^*(x', x'', \omega) dx'' \tag{15.334}$$

$$+ \int \underline{G}_p(x, x'', \omega) K_{p,p}^{\mathrm{fGR}}(x'', \omega) \underline{G}_p^*(x', x'', \omega) dx'' \tag{15.335}$$

$$+ \int G_{\mathrm{A}}(x, x'', \omega) K_{\mathrm{A}}^{\mathrm{RDF}}(x'', \omega) G_{\mathrm{A}}^*(x', x'', \omega) dx'' \tag{15.336}$$

$$+ \int G_{\mathrm{D}}(x, x'', \omega) K_{\mathrm{D}}^{\mathrm{RDF}}(x'', \omega) G_{\mathrm{D}}^*(x', x'', \omega) dx'' \tag{15.337}$$

where the K are the noise sources (see Section 15.3). The spatial coordinates x and x' each correspond to a contact.

For drift-diffusion simulations, $\underline{G}_n = \nabla G_n$. For hydrodynamic simulations, $\underline{G}_n$ is replaced by an effective Green function, $\underline{G}_n = \nabla G_n + G_{T_n} \nabla E_{\mathrm{C}} - \frac{5}{2} r_n k_{\mathrm{B}} T_n \nabla G_{T_n}$ (see Section 4.2.4 on page 15.130). Similar relations hold for $\underline{G}_p$.

The Green function G_{A} for random dopant fluctuations reads:

$$G_{\mathrm{A}} = G_n \frac{\partial \nabla \cdot \vec{J}_n}{\partial N_{\mathrm{A}}} + G_p \frac{\partial \nabla \cdot \vec{J}_p}{\partial N_{\mathrm{A}}} + G_{T_n} \frac{\partial (\nabla \cdot \vec{S}_n - \vec{J}_n \cdot \nabla E_{\mathrm{C}})}{\partial N_{\mathrm{A}}} + G_{T_p} \frac{\partial (\nabla \cdot \vec{S}_p - \vec{J}_p \cdot \nabla E_{\mathrm{V}})}{\partial N_{\mathrm{A}}} - G_{\Psi} \tag{15.338}$$

An analogous expression holds for G_D. The expression above is valid for hydrodynamic simulations. The expression for drift-diffusion simulations is similar. The derivatives with respect to N_A describe the impact of dopant fluctuations on mobility and band-gap narrowing. DESSIS takes them into account only if explicitly requested by the user (see Section 15.3.4 on page 15.294).

15.5 Noise output data

Several variables can be plotted during noise analysis. For each device, a `NoisePlot` section can be specified similar to the `Plot` section, where the data to be plotted is listed. Besides the standard data, additional noise-specific data or groups of data can be specified, as listed in Table 15.119 (for the device autocorrelation data) and Table 15.120 on page 15.297 (for the device cross-correlation data). In the tables, the abbreviations LNS (local noise source) and LNVSD (local noise voltage spectral density) are used.

Autocorrelation data refers to (Eq. 15.330) to (Eq. 15.336), when x and x' are identical. Data selected in the `NoisePlot` section is plotted for each device and observation node at a given frequency into a separate file. File names with the following format are used:

```
<noise-plot>_<device-name>_<ob-node>_<number>_acgf_des.dat
```

where `<noise-plot>` is the prefix specified by the `NoisePlot` option to `ACCoupled`.

Table 15.119 Device noise data for node autocorrelation

Keyword	Description	Reference equation
`eeDiffusionLNS`	Electron diffusion LNS	(Eq. 15.326)
`hhDiffusionLNS`	Hole diffusion LNS	
`eeMonopolarGRLNS`	Trace of electron monopolar GR LNS	(Eq. 15.327)
`hhMonopolarGRLNS`	Trace of hole monopolar GR LNS	
`eeFlickerGRLNS`	Trace of electron flicker GR LNS	(Eq. 15.328)
`hhFlickerGRLNS`	Trace of hole flicker GR LNS	
`LNVSD`	Total LNVSD	Sum of integrands in (Eq. 15.330)–(Eq. 15.336)
`eeLNVSD`	Total electron LNVSD	Sum of integrands in (Eq. 15.330)–(Eq. 15.332)
`hhLNVSD`	Total hole LNVSD	Sum of integrands in (Eq. 15.333)–(Eq. 15.335)
`eeDiffusionLNVSD`	Electron diffusion LNVSD	Integrand of (Eq. 15.330)
`hhDiffusionLNVSD`	Hole diffusion LNVSD	Integrand of (Eq. 15.333)
`eeMonopolarGRLNVSD`	Electron monopolar GR LNVSD	Integrand of (Eq. 15.331)
`hhMonopolarGRLNVSD`	Hole monopolar GR LNVSD	Integrand of (Eq. 15.334)
`eeFlickerGRLNVSD`	Electron flicker GR LNVSD	Integrand of (Eq. 15.332)
`hhFlickerGRLNVSD`	Hole flicker GR LNVSD	Integrand of (Eq. 15.335)

Table 15.119 Device noise data for node autocorrelation

Keyword	Description	Reference equation

`PoECReACGreenFunction`
`PoECImACGreenFunction`
`PoHCReACGreenFunction`
`PoHCImACGreenFunction`
	Real/imaginary (`Re`/`Im`) of G_n (`EC`) and G_p (`HC`)	

`Grad2PoECACGreenFunction` `Grad2PoHCACGreenFunction`	$\|\underline{G}_n\|^2$ and $\|\underline{G}_p\|^2$	
`AllLNS`	All used LNS	
`AllLNVSD`	All used LNVSD	
`GreenFunctions`	All used Green functions and their gradients	

In the case of $x \neq x'$ in (Eq. 15.330) to (Eq. 15.336), node cross-correlation spectra are computed and integrands become complex. Data specified in the `NoisePlot` section is plotted for each device and each pair of observation nodes at a given frequency into a separate file. The file names have the format:

```
<noise-plot>_<device-name>_<ob-node-1>_<ob-node-2>_<number>_acgf_des.dat
```

Table 15.120 Device noise data for node cross-correlation

Keyword	Description
`ReLNVXVSD` `ImLNVXVSD`	Real/imaginary parts of total cross LNVSD
`ReeeLNVXVSD` `ImeeLNVXVSD`	Real/imaginary parts of cross LNVSD for electrons
`RehhLNVXVSD` `ImhhLNVXVSD`	Real/imaginary parts of cross LNVSD for holes
`ReeeDiffusionLNVXVSD` `ImeeDiffusionLNVXVSD`	Real/imaginary parts of cross-diffusion LNVSD for electrons
`RehhDiffusionLNVXVSD` `ImhhDiffusionLNVXVSD`	Real/imaginary parts of cross-diffusion LNVSD for holes
`RehhMonopolarGRLNVXVSD` `ImhhMonopolarGRLNVXVSD`	Real/imaginary parts of cross-electron/hole diffusion LNVSD
`ReeeFlickerGRLNVXVSD ImeeFlickerGRLNVXVSD`	
`RehhFlickerGRLNVXVSD ImhhFlickerGRLNVXVSD`	
`AllLNVXVSD`	All use LNVXVSD

CHAPTER 16 Tunneling

16.1 Overview

In current microelectronic devices, tunneling has become a very important physical effect. In some devices, tunneling leads to undesired leakage currents (for gates in small MOSFETs). For other devices such as EEPROMs, tunneling is essential for the operation of the device. The importance of tunneling is reflected by the multitude of models DESSIS offers to simulate it. These models differ in both numeric efficiency and physical sophistication. This section gives an overview of the available models and their recommended applications.

The tunneling models discussed refer to processes at interfaces or contacts. They all describe transport of charge. Tunneling also plays a role for some of the generation–recombination models (see Chapter 9 on page 15.201). These models do not deal with spatial transport of charge and, therefore, are not discussed here. In addition to tunneling, hot carrier injection can also contribute to carrier transport across barriers. To model hot carrier injection, see Chapter 17 on page 15.317.

The most versatile tunneling model available in DESSIS is the nonlocal tunneling model (see Section 16.4 on page 15.306). It allows to describe tunneling at Schottky and gate contacts as well as tunneling at heterointerfaces and semiconductor–metal interfaces. This model:

- Handles arbitrary barrier shapes.
- Includes carrier heating terms.
- Allows to describe tunneling between valence band and conduction band.
- Optionally, uses nonparabolic two-band dispersion relation (important for valence to conduction band tunneling).
- By default, uses a simple WKB-based approximation for the tunneling probability.
- Optionally, accurately computes the tunneling probability using the Schrödinger equation.
- Typically increases the number of off-diagonal matrix elements significantly and, therefore, slows down the simulation.

It is recommended that this model is used to describe tunneling at Schottky contacts, tunneling in heterostructures, and gate leakage through thin, stacked insulators.

The second most powerful model is the Schenk direct tunneling model (see Section 16.3 on page 15.302). This model:

- Is based on the physically very accurate quantum mechanical transmission computation.
- Increases the number of off-diagonal matrix elements only marginally.
- Optionally, accounts for image charge effects (at the cost of reduced numeric robustness).
- Assumes a trapezoidal barrier (this restricts the range of application to tunneling through insulators).
- Neglects heating of the tunneling carriers.

It is recommended that this model is used to describe leakage through thin gate insulators, provided those are of uniform or of uniformly graded composition.

The simplest tunneling model is the Fowler–Nordheim model (see Section 16.2). Fowler–Nordheim tunneling is a special case of tunneling also covered by the nonlocal and Schenk tunneling models, where tunneling is to the conduction band of the oxide. The model is simple and efficient, and has proven useful to describe erase operations in EEPROMs, which is the application for which this model is recommended.

Gate current computation is important in steady state analysis (for example, gate current simulations and device degradation problems) and transient simulation, where write and erase simulations of EEPROM devices are crucial. If the simulated device contains a floating gate, gate currents are used to update the charge on the floating gate after each time step in transient simulations.

If EEPROM cells are simulated in 2D, it is generally necessary to include an additional coupling capacitance between the control and floating gates to account for the additional influence of the third dimension on the capacitance between these electrodes (see Section 4.5.1.6 on page 15.143). The additional floating gate capacitance can be specified as `FGcap` in the `Electrode` statement (see Section 2.3 on page 15.39).

16.2 Fowler–Nordheim tunneling

16.2.1 Syntax and implementation

To switch on the Fowler–Nordheim tunneling model, specify `Fowler` as an argument of the `GateCurrent` statement inside the `Physics` section, as follows:

```
Physics { GateCurrent(Fowler) }
```

or:

```
Physics { GateCurrent(Fowler(EVB)) }
```

The second specification activates the tunneling of electrons from and to the valence band as discussed in Section 16.2.2 on page 15.301.

If `GateCurrent` is specified in the bulk `Physics` statement in this way, DESSIS computes gate currents between all semiconductor–insulator interfaces and the nearest electrodes. If the gate current is to be computed only for selected interfaces and electrodes, specify the keyword `GateCurrent` inside a material-interface or region-interface `Physics` section (see Section 2.5.5 on page 15.49).

There is an additional option to restrict the tunneling current to a particular gate electrode by specifying the keyword `GateName` in the `GateCurrent` statement:

```
Physics(MaterialInterface="Silicon/Oxide") {
   GateCurrent( Fowler GateName="real_electrode_name" )
}
```

The Fowler–Nordheim tunneling model can be used with any of the hot carrier injection models (see Chapter 17 on page 15.317), for example:

```
GateCurrent( Fowler Lucky )
```

switches on the Fowler–Nordheim tunneling model and the classical lucky electron injection model simultaneously.

16.2.2 Model description

The tunneling of electrons through insulators in the presence of high electric fields is incorporated in DESSIS by the Fowler–Nordheim equation:

$$j_{FN} = AF^2 e^{-\frac{B}{F}} \tag{15.339}$$

where j_{FN} is the tunnel current density, F is the insulator electric field at the interface, and A and B are physical constants. The electric field at the interface shown by Tecplot-ISE is an interpolation of the fields in both regions, but DESSIS uses the insulator field internally.

Due to the large energy difference between oxide and silicon conduction bands, tunneling electrons create electron–hole pairs in the erase operation when they enter the semiconductor. This additional carrier generation is included by an energy-independent multiplication factor $\gamma > 1$:

$$j_n = \gamma \cdot j_{FN} \tag{15.340}$$

$$j_h = (\gamma - 1) \cdot j_{FN} \tag{15.341}$$

If the electrons flow in an opposite direction, by default, γ is equal to 1. The formulas above reflect the default behavior of DESSIS, but sometimes the Fowler–Nordheim equation is used to emulate other tunneling effects, for example, the tunneling of electrons from the valence band into the gate. Such capability is activated by the additional keyword `EVB` in the input file. DESSIS will continue to function even if $\gamma < 1$. For any electron tunneling direction, particularly a floating body SOI, this tunneling current is important because it strongly defines floating body potential. The tunneling current is implemented as a current boundary condition at interfaces. Different coefficients are needed for the write and erase operations because, in the first case, the electrons are emitted from monocrystalline silicon and, in the latter case, they are emitted from the polysilicon contact into the oxide.

16.2.3 Model parameters

The parameters of the Fowler–Nordheim model can be modified in the `FowlerModel` section of the parameter file. DESSIS uses two parameter sets (denoted as `erase` and `write`) depending on the direction of the electric field between the contact and semiconductor in the oxide layer. For example, if the field points from the contact to the semiconductor (that is, electrons flow into the contact), the 'write' parameter set is used.

Table 15.121 lists the keywords of the parameters and their default values. Calculated tunneling characteristics based on these parameters are plotted in Figure 15.65 on page 15.302.

Table 15.121 Coefficients for Fowler–Nordheim tunneling (defaults for silicon–oxide interface)

Symbol	Parameter name	Default value	Unit	Remarks
`A (erase)`	`Ae`	1.82×10^{-7}	A/V^2	`A` for the erase cycle
`B (erase)`	`Be`	1.88×10^{8}	V/cm	`B` for the erase cycle

Table 15.121 Coefficients for Fowler–Nordheim tunneling (defaults for silicon–oxide interface)

Symbol	Parameter name	Default value	Unit	Remarks
`A (write)`	`Aw`	1.23×10^{-6}	A/V^2	`A` for the write cycle
`B (write)`	`Bw`	2.37×10^{8}	V/cm	`B` for the write cycle
γ	`Gm`	1.0	1	

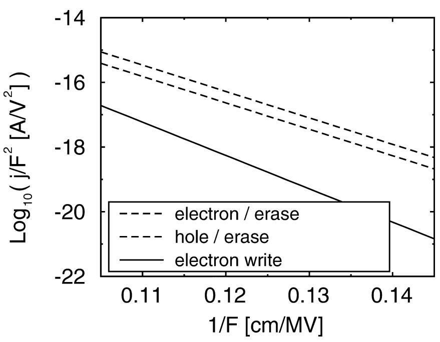


Figure 15.65 Fowler–Nordheim current densities for erase/write operations including multiplicative factor (default values used)

16.3 Direct tunneling through gate oxides

Direct tunneling is the main gate leakage mechanism for oxides thinner than 3 nm. It turns into Fowler–Nordheim tunneling at oxide fields higher than approximately 6 MV/cm, independent of the oxide thickness. When solving the continuity equations, the tunnel current obtained from the model is balanced with the drift-diffusion current in the semiconductor, resulting in a self-consistent solution.

In some application examples (n-channel MOSFETs), convergence problems can occur far from equilibrium if the image force effect is switched on, probably due to the nonlocal nature of the problem. With such possible convergence problems, the image force effect is switched off by default in DESSIS.

To switch on the image force effect, the parameters `E0`, `E1`, and `E2` must be specified (in eV) in the parameter file. If these values are all equal, the image force is neglected. Recommended values for both electrons and holes are `E0=0`, `E1=0.3`, and `E2=0.7`. For most purposes, this effect can be reasonably modeled as an effective lowering of the trapezoidal barrier (by adjustment of the input parameter E_B). If the minority carrier substrate current is negligible, it is recommended that minority carriers are excluded from the coupled solve.

16.3.1 Syntax and implementation

Like other gate current mechanisms, direct tunneling is specified as an option of the `GateCurrent` statement (see Section 16.2.1 on page 15.300) in an appropriate interface `Physics` section:

```
Physics(MaterialInterface="Silicon/Oxide") {
   GateCurrent( DirectTunneling )
}
```

The keyword `GateName` and the compatibility with the hot carrier injection models (see Chapter 17 on page 15.317) are as for the Fowler–Nordheim model, see Section 16.2 on page 15.300.

To plot the direct tunneling current, the keywords `eDirectTunnel` or `hDirectTunnel` must be included in the `Plot` section:

```
Plot {eDirectTunnel hDirectTunnel}
```

16.3.2 Model description

This section contains a short description of the model, which has been fully described elsewhere [118]. Here, the formulas are for electrons only, but the implementation is performed for both electrons and holes, assuming that the potential barrier for hole tunneling is determined by the SiO_2 valence band edge. The actual barrier for hole tunneling is still uncertain as it is poorly determined from experimental data. The density of the elastic tunnel current is given by:

$$j_n = \frac{qm_c^*k_B}{2\pi^2h^3}\int_0^\infty dE\Upsilon(E)\left\{T(0)\ln\left(\exp\left[\frac{E_{F,S}(0)-E_c(0)-E}{k_BT(0)}\right]+1\right)\right. \\ \left.-T(\mathrm{d})\ln\left(\exp\left[\frac{E_{F,M}(\mathrm{d})-E_c(0)-E}{k_BT(\mathrm{d})}\right]+1\right)\right\} \tag{15.342}$$

where d is the effective thickness of the barrier, q is the elementary charge, m_c^* is a mass prefactor, k_B is the Boltzmann constant, $E_{F,S}(0)$ is the substrate Fermi energy at the Si–SiO_2 interface, $E_{F,M}(\mathrm{d})$ is the gate Fermi energy at the gate–SiO_2 interface, $E_c(0)$ is the conduction band energy at the Si–SiO_2 interface, E is the energy of the elastic tunnel process (relative to the zero point $E_c(0)$), T is the temperature, and $\Upsilon(E)$ is the quantum-mechanical transmission coefficient. For the latter, the explicitly available expression for a trapezoidal potential barrier is used:

$$\Upsilon(E) = \frac{2}{1+g(E)} \tag{15.343}$$

with:

$$g(E) = \frac{\pi^2}{2}\left\{\sqrt{\frac{E_M}{E}}\sqrt{\frac{m_c}{m_M}}(Bi'_dAi_0-Ai'_dBi_0)^2+\right. \\ \sqrt{\frac{E}{E_M}}\sqrt{\frac{m_M}{m_c}}(Bi_dAi'_0-Ai_dBi'_0)^2+ \\ \frac{\sqrt{m_Mm_{\mathrm{Si}}}}{m_{\mathrm{ox}}}\frac{h\Theta_{\mathrm{ox}}}{\sqrt{EE_M}}(Bi'_dAi'_0-Ai'_dBi'_0)^2+ \\ \left.\frac{m_{\mathrm{ox}}}{\sqrt{m_Mm_{\mathrm{Si}}}}\frac{\sqrt{EE_M}}{h\Theta_{\mathrm{ox}}}(Bi_dAi_0-Ai_dBi_0)\right\} \tag{15.344}$$

and:

$$Ai_0 \equiv Ai\left(\frac{E_B(E)-E}{h\Theta_{\mathrm{ox}}}\right),\quad Ai_d \equiv Ai\left(\frac{E_B(E)-qF_{\mathrm{ox}}d-E}{h\Theta_{\mathrm{ox}}}\right) \tag{15.345}$$

and so on, where $h\Theta_{\mathrm{ox}} = (q^2h^2F_{\mathrm{ox}}^2/2m_{\mathrm{ox}})^{1/3}$ and $E_B(E)$ denotes the (silicon-side) barrier height for electrons, which is modeled as a function of the tunnel energy E. E_M is the tunneling energy with respect to the conduction band edge in the contact $E_M = E-E_c(d)-E_c(0)$.

For compatibility with an earlier implementation of the tunneling model, E_M is truncated to the value specified by the parameter `E_F_M` if it exceeds this value. $F_{\mathrm{ox}} = V_{\mathrm{ox}}/d$ is the electric field in the oxide (assumed to be uniform within the oxide, and including a band edge–related term when different barrier heights are specified at the two insulator interfaces). The quantities m_M, m_{ox} and m_{Si} represent the effective electron masses in the three materials, respectively. Ai and Bi are Airy functions. Ai' and Bi' are the first derivatives of the Airy functions.

To solve this set of equations, the insulator electric field F_{ox} and semiconductor Fermi level $E_{F,S}(0)$ are needed. Outside of thermal equilibrium, the Fermi level is split into the electron and hole quasi-Fermi levels, ϕ_n, ϕ_p, such that $E_{F,S}(0) = \phi_n$ is used for electron tunneling and $E_{F,S}(0) = \phi_p$ is used for hole tunneling, respectively. F_{ox} is obtained by solving the Laplace equation in the insulator.

The tunnel current obtained from (Eq. 15.342) is balanced with the drift-diffusion current in the semiconductor when solving the continuity equations, resulting in a self-consistent solution.

The expression for the tunneling transmission coefficient is based on these assumptions:

- Independent particle approximation (all scattering processes and interaction with the environment are implicitly absorbed by the one-particle, quasi-equilibrium, distribution functions and by the potential model for $\Upsilon(E)$)
- Plane waves for incoming and outgoing states
- No fixed oxide charges
- Parabolic E_k relations (energy bands) in all three materials

Further, the explicit dependence of $\Upsilon(E)$ on the energy that is perpendicular to the tunnel current is neglected. With this approximation, an additional numeric integration is avoided.

16.3.2.1 Image force effect

Ultrathin oxide barriers are affected by the image force effect. If the latter is neglected, $E_B(E)$ is the bare barrier height E_B, which is an input parameter and assumed to be known from tables or measurements. The image force effect is included in the model by taking $E_B(E)$ as an energy-dependent pseudobarrier:

$$E_B(E) = E_B(E_0) + \frac{E_B(E_2) - E_B(E_0)}{(E_2 - E_0)(E_1 - E_2)}(E - E_0)(E_1 - E) - \frac{E_B(E_1) - E_B(E_0)}{(E_1 - E_0)(E_1 - E_2)}(E - E_0)(E_2 - E) \tag{15.346}$$

where the three energy values $E_j (j = 0, 1, 2)$ are chosen in the lower energy range of the barrier potential (between 0 and 1.5eV in practical cases). If these values are chosen to be equal, the image force effect is automatically switched off. Otherwise, a numeric iteration of the equation:

$$S_{\mathrm{tra}}(E) = S_{\mathrm{im}}(E) \tag{15.347}$$

at three discrete energies $E_j (j = 0, 1, 2)$ is performed for each bias point, which results in three pseudobarrier heights $E_B(E_j) (j = 0, 1, 2)$ of (Eq. 15.346). In (Eq. 15.347), $S_{\mathrm{tra}}(E)$ is the action of the trapezoidal pseudobarrier:

$$S_{\mathrm{tra}}(E) = \frac{2}{3}\left|\left[\frac{E_B(E) - qF_{\mathrm{ox}}d - E}{h\Theta_{\mathrm{ox}}}\right]^{\frac{3}{2}}\Theta[E_B(E) - qF_{\mathrm{ox}}d - E] - \left[\frac{E_B(E) - E}{h\Theta_{\mathrm{ox}}}\right]^{\frac{3}{2}}\right| \tag{15.348}$$

and $S_{\mathrm{im}}(E)$ is the action of the respective image potential barrier:

$$S_{\mathrm{im}}(E) = \sqrt{\frac{2m_{\mathrm{ox}}}{h^2}} \int_{x_l(E)}^{x_r(E)} d\xi \sqrt{E_B - qF_{\mathrm{ox}}\xi + E_{\mathrm{im}}(\xi) - E} \tag{15.349}$$

(for example, $E_B = \mathrm{const} = 3.15\mathrm{eV}$ for Al-SiO$_2$).

$x_{l,r}(E)$ denotes the classical turning points for a given carrier energy that follow from:

$$E_B - qF_{\mathrm{ox}}x_{l,r} + E_{\mathrm{im}}(x_{l,r}) = E \tag{15.350}$$

For the thickness d of the pseudobarrier, $d = x_r(E{=}0) - x_l(E{=}0)$ is used, that is, the distance between the two turning points at the energy of the semiconductor conduction band edge of the interface. All energies are relative to that value.

Therefore, d is smaller than the user-given oxide thickness, when the image force effect is considered. The image potential itself is given by:

$$E_{\mathrm{im}}(x) = \frac{q^2}{16\pi\varepsilon_{\mathrm{ox}}} \sum_{n=0}^{10} (k_1k_2)^n \left[\frac{k_1}{nd+x} + \frac{k_2}{d(n+1)-x} + \frac{2k_1k_2}{d(n+1)}\right] \tag{15.351}$$

with:

$$k_1 = \frac{\varepsilon_{\mathrm{ox}} - \varepsilon_{\mathrm{Si}}}{\varepsilon_{\mathrm{ox}} + \varepsilon_{\mathrm{Si}}}, \quad k_2 = \frac{\varepsilon_{\mathrm{ox}} - \varepsilon_{\mathrm{M}}}{\varepsilon_{\mathrm{ox}} + \varepsilon_{\mathrm{M}}} = -1 \tag{15.352}$$

In (Eq. 15.352), $\varepsilon_{\mathrm{ox}}$ is the dielectric permittivity of the gate oxide.

In the special case of $V_{\mathrm{ox}} = F_{\mathrm{ox}} = 0$, the arguments of the Airy functions diverge. Then, $g(E)$ becomes:

$$g(E) = \frac{1}{2}\left\{\left(\sqrt{\frac{E_{F,M}}{E}}\sqrt{\frac{m_c}{m_M}} + \sqrt{\frac{E}{E_{F,M}}}\sqrt{\frac{m_M}{m_c}}\right)\cos^2(d\kappa) + \right.$$
$$\left.\left(\frac{\sqrt{m_M m_{\mathrm{Si}}}}{m_{\mathrm{ox}}}\frac{|E - E_B(E)|}{\sqrt{EE_{F,M}}} + \frac{m_{\mathrm{ox}}}{\sqrt{m_M m_{\mathrm{Si}}}}\frac{\sqrt{EE_{F,M}}}{|E - E_B(E)|}\right)\sin^2(d\kappa)\right\} \tag{15.353}$$

with $\kappa = \sqrt{2m_{\mathrm{ox}}[E - E_B(E)]}/h$.

16.3.3 Model parameters

The parameters of the direct tunneling model are modified in the `DirectTunneling` section of the parameter file. The appropriate default parameters for an oxide barrier on silicon are in Table 15.122. The parameters are specified according to the interface.

Table 15.122 Coefficients for direct tunneling (defaults for oxide barrier on silicon)

Symbol	Parameter name	Electrons	Holes	Unit	Description
$\varepsilon_{\mathrm{ox}}$	`eps_ins`	2.13	2.13	1	Optical dielectric constant
$E_{F,M}$(d)	`E_F_M`	11.7	11.7	eV	Metal Fermi energy

Table 15.122 Coefficients for direct tunneling (defaults for oxide barrier on silicon)

Symbol	Parameter name	Electrons	Holes	Unit	Description
m_M	`m_M`	1	1	m0	Effective mass in metal gate
m_{ox}	`m_ins`	0.50	0.77	m0	Effective mass in insulator
m_{Si}	`m_s`	0.19	0.16	m0	Effective mass in semiconductor
$\mathrm{m_c}$	`m_dos`	0.32	0	m0	Semiconductor DOS effective mass
E_B	`E_barrier`	3.15	4.73	eV	Semiconductor/insulator barrier (no image force)
$\mathrm{E_0, E_1, E_2}$	`E0`, `E1`, `E2`	0.0 / 0.0		eV	Energy nodes 0, 1, and 2 for pseudobarrier calculation

If tunneling occurs between two semiconductors, the coefficients for metal (`m_M` and `E_F_M`) are not used. The semiconductor parameters for the respective interface are used. It is not valid to have contradicting parameter sets for two interfaces of an insulator between which tunneling occurs. In particular, `eps_ins`, `m_ins`, `E0`, `E1`, and `E2` must agree in the parameter specifications for both interfaces.

16.4 Nonlocal tunneling at interfaces and contacts

The magnitude of the tunneling current depends on the band edge profile along the entire path between the points connected by tunneling. This makes tunneling a nonlocal process. In general, the band edge profile has a complicated shape, and DESSIS must compute it by solving the transport equations and the Poisson equation. The model described here takes this dependence fully into account. It provides good convergence and consistent results for AC analysis by including the nonlocal couplings introduced by tunneling into the Jacobian of the system. The model can be applied to contacts (such as Schottky contacts and gate contacts) and interfaces (such as heterointerfaces and poly/oxide interfaces).

To use the nonlocal tunneling model:

1. Construct a special purpose 'nonlocal' mesh (see Section 16.4.1).
2. Specify the physical details of the tunneling model (see Section 16.4.2 on page 15.307).
3. Adjust the physical and numeric parameters (see Section 16.4.3 on page 15.309).

16.4.1 Defining a nonlocal mesh

It is necessary to specify a special purpose 'nonlocal' mesh for each interface or contact for which you want to use the nonlocal tunneling model. The nonlocal mesh connects the mesh vertices in the vicinity of the contact or interface to the closest point on it. These connections form the tunneling paths for the carriers.

NOTE If the nonlocal tunneling model is switched on as described in Section 16.4.2, but the specification for the nonlocal mesh as described here is omitted, DESSIS will construct a default nonlocal mesh. The default nonlocal mesh feature provides compatibility with previous DESSIS versions. However, it is unlikely to fit the needs of your particular situation. Therefore, do not rely on the default nonlocal mesh.

To control the construction of the nonlocal mesh, use the options of the keyword `NonLocal`. For basic use, specify `NonLocal` in the contact-specific or interface-specific `Math` section, with the option `Length`. `Length` (given in centimeters) is the maximum tunneling distance and, therefore, determines for which vertices DESSIS must construct nonlocal mesh lines.

You must select `Length` to be at least as large as the thickness of the tunneling barrier. For example, with:

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

DESSIS constructs nonlocal mesh lines for vertices up to a distance of 5 nm from the `Gate` electrode.

NOTE The areas covered by the nonlocal meshes for different interfaces or contacts must not overlap.

The prohibition of overlap means that you must specify a nonlocal mesh only for one side of a tunneling barrier. If on one side of the tunneling barrier there is a contact or a metal–nonmetal interface, specify the mesh for this contact or interface. In other cases, it is recommended to specify the nonlocal mesh for the interface on the side of the barrier from which the carriers will mainly tunnel away. For tunneling barriers consisting of multiple layers, do not specify a nonlocal mesh for the interfaces internal to the barrier.

For more options to the keyword `NonLocal`, see Table 15.36 on page 15.84. For details about the nonlocal mesh and its construction, see Section 2.10.7.4 on page 15.85.

NOTE The nonlocality of tunneling can increase dramatically the time that is needed to solve the linear systems in the Newton iteration. To limit the speed degradation, it is important to choose a `Length` that is not larger than that physically needed. If the performance is still unacceptable, optimize the construction of the nonlocal mesh using the advanced options of `NonLocal` (see Table 15.36 and Section 2.10.7.4).

16.4.2 Specifying the physical model

The physical aspects of the nonlocal tunneling model are activated and controlled in an electrode-specific or interface-specific `Physics` section. DESSIS distinguishes two components of the tunneling current to an interface or a contact: The tunneling current that goes to the conduction band at the interface or contact, and the tunneling current that goes to the valence band. To switch on the former, specify the keyword `eBarrierTunneling`. To switch on the latter, specify the keyword `hBarrierTunneling`. These keywords are options to `Recombination`. For example:

```
Physics(Electrode="Gate"){
   Recombination(eBarrierTunneling hBarrierTunneling)
}
```

switches on electron and hole tunneling to the electrode `Gate`, and:

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

switches on tunneling to the conduction band of silicon at all silicon-oxide interfaces. The option `PeltierHeat` to `eBarrierTunneling` activates the Peltier heating of the tunneling particles.

NOTE The recommendations in Section 16.4.1 on page 15.306, on the proper choice of the contacts and interfaces for which to construct a nonlocal mesh, also apply to the activation of the nonlocal tunneling model.

By default, all tunneling to the conduction band at the interface or contact originates from the conduction band in the bulk, $j_C = j_{CC}$ ('electron tunneling'), see (Eq. 15.362). Likewise, by default, the tunneling to the valence band at the interface or contact originates from the valence band in the bulk, $j_V = j_{VV}$ (that is, 'hole tunneling').

In addition, DESSIS supports nonlocal band-to-band tunneling. To include the contributions by tunneling to and from the valence band in the bulk in j_C, specify `eBarrierTunneling` with the `Band2Band` option. Then, $j_C = j_{CC} + j_{CV}$, see (Eq. 15.362) and (Eq. 15.364). To include contributions by tunneling to and from the conduction band in the bulk to j_V, specify `hBarrierTunneling` with the `Band2Band` option. Then, $j_V = j_{VC} + j_{VV}$.

Figure 15.66 illustrates the four contributions to the total tunneling current.

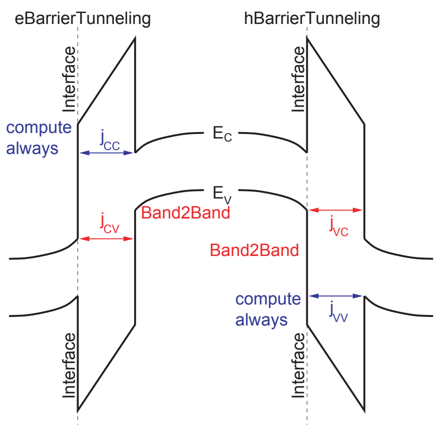


Figure 15.66 Various nonlocal tunneling current contributions

By default, DESSIS assumes a single-band parabolic band structure for the tunneling particles, uses a WKB-based model for the tunneling probability, and ignores band-to-band tunneling and Peltier heating. Options to `eBarrierTunneling` and `hBarrierTunneling` override the default behavior. For available options, see Table 15.123. For a detailed discussion of the physics of the nonlocal tunneling model, see Section 16.4.5 on page 15.310.

Table 15.123 Physical model options for eBarrierTunneling and hBarrierTunneling

Keyword	Description
`Band2Band`	Includes tunneling of valence band electrons to the conduction band and of conduction band electrons to the valence band (default is `-Band2Band`) (see (Eq. 15.363)).
`BandGap`	Allows tunneling into the band gap at the interface for which tunneling is specified (default is `-BandGap`). This option provides backward compatibility with previous DESSIS versions.
`PeltierHeat`	Includes the Peltier heating terms for tunneling carriers (default is `-PeltierHeat`) (see (Eq. 15.367) and (Eq. 15.368)).
`Schroedinger`	Activates a Schrödinger equation–based model for tunneling probabilities (see Section 16.4.5.2) instead of the WKB-based model used by default (see Section 16.4.5.1). This option does not work with the `TwoBand` or `Band2Band` option.

Table 15.123 Physical model options for eBarrierTunneling and hBarrierTunneling

Keyword	Description
`Transmission`	Activates additional interface transmission coefficients according to (Eq. 15.359).
`TwoBand`	Switches to a two-band dispersion relation (default is `-TwoBand`) (see (Eq. 15.358)).

16.4.3 Physical and numeric parameters

The model provides four distinct fit parameters. The dimensionless prefactors g_C and g_V (see (Eq. 15.361) and (Eq. 15.363)) for tunneling to the conduction band or valence band are specific to the contact or interface to which the tunneling model is applied. Therefore, specify them in the section of the parameter file specific to this contact or interface. Both prefactors default to 1.

The effective tunneling masses for the conduction band and valence band (m_C* and m_V*, see (Eq. 15.354) and (Eq. 15.355)) are properties of the materials that form the tunneling barrier. Hence, specify them in the region- or material-specific sections of the parameter file.

DESSIS treats effective tunneling masses of value zero as undefined. If an effective tunneling mass for a region is undefined, DESSIS uses the effective tunneling mass for the interface or contact for which tunneling is activated. If this parameter is undefined also, DESSIS uses a built-in default of 0.1. By default, all effective tunneling masses are zero (that is, undefined) and, therefore, DESSIS substitutes a value of 0.1 everywhere. This feature ensures backward compatibility with earlier DESSIS versions where the effective tunneling masses were interface or contact parameters.

For example:

```
MaterialInterface="Silicon/Oxide" {
   BarrierTunneling {
      g = 1 , 2
   }
}
Material = "Oxide" {
   BarrierTunneling {
      mt = 0.42 , 1.0
   }
}
```

changes the prefactors g_C and g_V for silicon-oxide interfaces to 1 and 2, respectively. The example also changes the effective tunneling masses m_C* and m_V* in oxide to 0.42 and 1.

Specify numeric parameters for the model in the `Math` section specific to the interface or contact to which the nonlocal tunneling model is applied. The `EnergyResolution(NonLocal)` parameter (given in eV) is a lower limit for the energy step that DESSIS uses to perform integrations over band-edge energies.

If `EnergyResolution(NonLocal)` has a negative value, DESSIS approximates integrals like in (Eq. 15.369) by a single energy value. Additionally, DESSIS changes other computational details to provide compatibility with a numerically less accurate model implemented in an earlier DESSIS version.

The parameter `Digits(Nonlocal)` determines the relative accuracy (the number of valid decimal digits) to which DESSIS computes the tunneling currents. The default value for `Digits(Nonlocal)` is 2, whereas for `EnergyResolution(NonLocal)`, it is 0.005. For example:

```
Math(Electrode="Gate") {
   Digits(NonLocal)=3
   EnergyResolution(NonLocal)=0.001
   Nonlocal(Length=1e-7)
}
```

increases the energy resolution for tunneling at the `Gate` contact to 1 meV and the relative accuracy of the tunneling current computation to three digits. Additionally, this example contains the nonlocal mesh specification (see Section 16.4.1 on page 15.306 and Section 2.10.7.4 on page 15.85).

16.4.4 Visualizing nonlocal tunneling

To visualize nonlocal tunneling, specify the keyword `eBarrierTunneling` or `hBarrierTunneling` in the `Plot` section (see Section 2.6 on page 15.52). The quantities plotted are in units of $\mathrm{cm}^{-3}\mathrm{s}^{-1}$ and represent the rate at which electrons and holes are generated or disappear due to tunneling.

The rates are plotted vertexwise and are averaged over the semiconductor volume controlled by a vertex. Therefore, they depend on the mesh spacing. This dependence can become particularly strong at interfaces where the band edges change abruptly.

16.4.5 Physics of nonlocal tunneling model

The nonlocal tunneling model in DESSIS is based on the approach presented in the literature [128], but provides significant enhancements over the model presented there.

16.4.5.1 WKB tunneling probability

The computation of the tunneling probabilities Γ_C (for carriers tunneling to the conduction band at the interface or contact) and Γ_V (for tunneling to the valence band) is, by default, based on the WKB approximation. The WKB approximation uses the local (imaginary) wave numbers of particles at position r and with energy ε:

$$\kappa_C(r,\varepsilon) = \sqrt{2m_0 m_C^*(r)\left|E_C(r)-\varepsilon\right|}\,\Theta[E_C(r)-\varepsilon]/\hbar \tag{15.354}$$

$$\kappa_V(r,\varepsilon) = \sqrt{2m_0 m_V^*(r)\left|\varepsilon-E_V(r)\right|}\,\Theta[\varepsilon-E_V(r)]/\hbar \tag{15.355}$$

Here, κ_C is an approximation based on a parabolic extension of the conduction band structure into the gap, m_C^* is the effective conduction band tunneling mass, and E_C is the conduction band energy. Likewise, κ_V is an approximation based on the valence band structure, m_V^* is the effective valence band tunneling mass, and E_V is the valence band energy. m_0 is the free electron mass and $\hbar$ is the reduced Planck constant. Both effective tunneling masses are adjustable parameters (see Section 16.4.3 on page 15.309).

Using the local wave numbers and the interface reflection coefficients T_{CC} and T_{VV}, the tunneling probability between position 0 and r for a particle with energy ε can be written as:

$$\Gamma_{CC}(r, \varepsilon) = T_{CC}(0, \varepsilon)\exp\left(-2\int_0^r \kappa_C(r', \varepsilon)dr'\right)T_{CC}(r, \varepsilon) \tag{15.356}$$

and:

$$\Gamma_{VV}(r, \varepsilon) = T_{VV}(0, \varepsilon)\exp\left(-2\int_0^r \kappa_V(r', \varepsilon)dr'\right)T_{VV}(0, \varepsilon) \tag{15.357}$$

However, if the option `TwoBand` is specified to `eBarrierTunneling` (see Table 15.123 on page 15.308), DESSIS replaces κ_C in (Eq. 15.356) with the two-band dispersion relation:

$$\kappa = \frac{\kappa_C\kappa_V}{\sqrt{\kappa_C^2 + \kappa_V^2}} \tag{15.358}$$

If the option `TwoBand` is specified to `hBarrierTunneling`, DESSIS replaces κ_V in (Eq. 15.357) with the two-band relation (Eq. 15.358). Near the conduction and the valence band edge, the two-band dispersion relation approaches the single band dispersion relations (Eq. 15.354) and (Eq. 15.355), and provides a smooth interpolation in between. Figure 15.67 illustrates this.

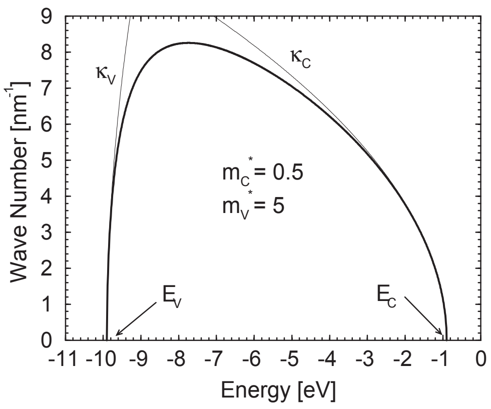


Figure 15.67 Comparison of two-band and single-band dispersion relations

The two-band dispersion relation does not distinguish between electrons and holes. In particular, for the two-band dispersion relation, $\Gamma_{CC} = \Gamma_{VV}$.

The two-band dispersion relation is most useful when band-to-band tunneling is active (keyword `Band2Band`, see Table 15.123). However, the two-band dispersion relation can be used independently from band-to-band tunneling.

By default, the interface transmission coefficients T_{CC} and T_{VV} in (Eq. 15.356) and (Eq. 15.357) equal one. If the `Transmission` option is specified to `eBarrierTunneling` [178]:

$$T_{CC}(x, \varepsilon) = \frac{v_-(x, \varepsilon)\sqrt{v_-(x, \varepsilon)^2 + 16v_+(x, \varepsilon)^2}}{v_+(x, \varepsilon)^2 + v_-(x, \varepsilon)^2} \tag{15.359}$$

Here, $v_-(x, \varepsilon)$ denotes the velocity of a particle with energy ε on the side of the interface or contact at position x where the particle moves freely in the conduction band, and $v_+(x, \varepsilon)$ denotes the imaginary velocity on the side of the tunneling barrier (where the particle is in the gap). If the particle is free or in the barrier on both sides, $T_{\mathrm{CC}}(x, \varepsilon) = 1$. Velocities are the derivatives of the particle energy with respect to the wave number, $v = |\partial\varepsilon / \partial\hbar\kappa_{\mathrm{C}}|$. With the `TwoBand` option, $v_+ = |\partial\varepsilon / \partial\hbar\kappa|$ [177] (see (Eq. 15.358)). If the `Transmission` option is specified to `hBarrierTunneling`, analogous expressions hold for T_{VV}.

By default, the band-to-band tunneling probabilities Γ_{CV} and Γ_{VC} are also given by the expressions (Eq. 15.356) and (Eq. 15.357), respectively. When the `TwoBand` and `Transmission` options are specified for `eBarrierTunneling` to compute Γ_{CV}, v_- in the expression for $T_{\mathrm{CC}}(r, \varepsilon)$ (see (Eq. 15.359)) is the velocity of the particle in the valence band and computed from valence band parameters. The converse holds for Γ_{VC} when those options are specified for `hBarrierTunneling`.

For metals and contacts, the band-edge energy to compute the interface transmission coefficients is obtained from the `FermiEnergy` parameter of the `BandGap` parameter set for the metal or contact. The masses used to compute the velocities are the tunneling effective masses for the metal or contact.

16.4.5.2 Schrödinger equation–based tunneling probability

In addition to the WKB-based models discussed in Section 16.4.5.1 on page 15.310, DESSIS can compute tunneling probabilities based on the Schrödinger equation. For the Schrödinger equation–based model, DESSIS computes the tunneling probability $\Gamma_{\mathrm{CC}}(E)$ (or $\Gamma_{VV}(E)$) by solving the 1D Schrödinger equation:

$$\left(-\frac{d}{dr}\frac{1}{m(r)}\frac{d}{dr} + E_{\mathrm{C}}(r)\right)\Psi(r) = E(r) \tag{15.360}$$

between the classical turning points that belong to tunneling energy E. For $m(r)$, DESSIS uses the tunneling mass `mt` (see Section 16.4.3 on page 15.309).

For boundary conditions, DESSIS assumes incident and reflected plane waves outside the barrier on one side, and an evanescent plane wave on the other side. The energy for the plane waves is the greater of $k_{\mathrm{B}}T_n$ and $E - E_{\mathrm{C}}$, where the carrier temperature T_n and band-edge energy E_{C} are taken at the point immediately outside the barrier on the respective side.The masses outside the barrier are the tunneling masses `mt` that are valid there.

16.4.5.3 Nonlocal tunneling current

For an interface located at 0 and a point at $r > 0$, the expression for the net conduction band electron recombination rate due to tunneling to and from the conduction band at point 0^- immediately to the left of the interface is:

$$R_{\mathrm{CC}}(r) - G_{\mathrm{CC}}(r) = \Theta[\varepsilon - E_{\mathrm{C}}(0^-)]\frac{A^*_{\mathrm{CC}}}{qk_{\mathrm{B}}}\left|\frac{dE_{\mathrm{C}}}{dr}(r)\right|\Theta\left[-\frac{dE_{\mathrm{C}}}{dr}(r)\right]\Gamma_{\mathrm{CC}}(r, \varepsilon) \times \\ \left[T_n(r)\log\left(1 + \exp\left[\frac{E_{\mathrm{F}_n}(r) - \varepsilon}{k_{\mathrm{B}}T_n(r)}\right]\right) - T_n(0^-)\log\left(1 + \exp\left[\frac{E_{\mathrm{F}_n}(0^-) - \varepsilon}{k_{\mathrm{B}}T_n(0^-)}\right]\right)\right] \tag{15.361}$$

where E_{C} and E_{F_n} are the (position-dependent) conduction band edge and electron Fermi energy, $\varepsilon = E_{\mathrm{C}}(r)$, Θ is the unit step function, $A^*_{\mathrm{CC}} = g_{\mathrm{C}} \cdot A_0$ is the effective Richardson constant (with A_0 the Richardson constant for free electrons), g_C is a fit parameter (see Section 16.4.3 on page 15.309), and T_n is the electron temperature. R_{CC} relates to the first term and G_{CC} to the second term in the second line of (Eq. 15.361). Γ_{CC} is the tunneling probability (see (Eq. 15.356)). For contacts, metals, or in presence of the option `BandGap`, the unit step function directly to the right of the equation sign in (Eq. 15.361) is omitted.

The current density of electrons that tunnel from the conduction band in the bulk to the conduction band at an interface or a contact is the integral over the recombination rate (Eq. 15.361):

$$j_{CC} = -q\int_{0^+}^{\infty}[R_{CC}(r) - G_{CC}(r)]dr \tag{15.362}$$

Here, 0^+ denotes the location infinitesimally to the right of the interface.

The options for `hBarrierTunneling` and the expressions for the valence band to valence band tunneling current density j_{VV} are analogous.

16.4.5.4 Band-to-band contributions to the nonlocal tunneling current

If DESSIS finds the option `Band2Band` to `eBarrierTunneling` (see Table 15.123 on page 15.308), the total current that flows to the conduction band of the interface or contact includes also electrons that originate from the valence band at position r. The recombination rate of valence band electrons at r (in other words, the generation rate of holes at r) is:

$$R_{CV}(r) - G_{CV}(r) = \Theta[\varepsilon - E_C(0^-)]\frac{A_{CV}^*}{qk_B}\left|\frac{dE_V}{dr}(r)\right|\Theta\left[\frac{dE_V}{dr}(r)\right]\Gamma_{CV}(r,\varepsilon)\times \left[T_p(r)\log\left(1+\exp\left[\frac{E_{F_p}(r)-\varepsilon}{k_BT_p(r)}\right]\right) - T_n(0^-)\log\left(1+\exp\left[\frac{E_{F_n}(0^-)-\varepsilon}{k_BT_n(0^-)}\right]\right)\right] \tag{15.363}$$

where E_V and E_{F_p} are the valence band edge and the hole Fermi energy, $\varepsilon = E_V(r)$ and $A_{CV}^* = \sqrt{g_C \cdot g_V} \cdot A_0$. The prefactor g_V is a fit parameter, see Section 16.4.3. T_p is the hole temperature and the other symbols have the same meaning as in (Eq. 15.361). Γ_{CV} is the band-to-band tunneling probability discussed above.

Unless you use the `Band2Band` option to `eBarrierTunneling` (see Table 15.123), DESSIS assumes that $R_{CV}(r) - G_{CV}(r)$ vanishes. The modifications for contacts, metals, and the `BandGap` option are as for (Eq. 15.361).

The current density of electrons that tunnel from the valence band of the bulk to the conduction band at an interface or a contact is the integral over the recombination rate (Eq. 15.363):

$$j_{CV} = -q\int_{0^+}^{\infty}[R_{CV}(r) - G_{CV}(r)]dr \tag{15.364}$$

For band-to-band tunneling processes, the energy of the tunneling particles often lies deep in the gap of the barrier. The single-band dispersion relations that DESSIS uses by default (see (Eq. 15.354) and (Eq. 15.355)) are based on the band structure near the band edges, and may not be useful in this regime. The two-band dispersion relation according (Eq. 15.358) is a better choice. To use the two-band dispersion relation, specify the option `TwoBand` to `eBarrierTunneling` (see Table 15.123).

The options for `hBarrierTunneling` and the expressions for the conduction band to valence band tunneling current density j_{VC} are analogous.

16.4.5.5 Carrier heating

In hydrodynamic simulations, carrier transport leads to energy transport and, therefore, to heating or cooling of electrons and holes. The energy transport has a convective and a Peltier part. By default, DESSIS ignores the Peltier part. To include the Peltier terms for the tunneling particles, specify the option `PeltierHeat` to `eBarrierTunneling` or `hBarrierTunneling` (see Table 15.123 on page 15.308).

We approximate the convective part of the heat generation in the conduction and valence band at position r due to tunneling to and from the conduction band at the interface or contact as:

$$H_{\mathrm{conv,CC}}(r) = \frac{\delta}{2}k_{\mathrm{B}}[G_{\mathrm{CC}}(r)T_n(0^-) - R_{\mathrm{CC}}(r)T_n(r)] \tag{15.365}$$

and:

$$H_{\mathrm{conv,CV}}(r) = \frac{\delta}{2}k_{\mathrm{B}}[R_{\mathrm{CV}}(r)T_p(r) - G_{\mathrm{CV}}(r)T_n(0^-)] \tag{15.366}$$

By default, DESSIS neglects Peltier heating and uses $\delta = 3$, which corresponds to the three degrees of freedom of the carriers. If Peltier heating is included in a simulation, the convective contribution due to one degree of freedom is already contained in the Peltier heating term. Therefore, in this case, DESSIS uses $\delta = 2$. The convective parts of the heat flux to the conduction band at the interface or contact, due to tunneling from the conduction band and the valence band in the bulk, are the integrals of $-H_{\mathrm{conv,CC}}$ and $-H_{\mathrm{conv,CV}}$ over the positive r-axis. The expressions for the convective part of the heat flux to the valence band at the interface or contact are analogous.

If the computation of Peltier heating is activated (see Table 15.123), DESSIS computes additional heating terms.

The Peltier part of the heat flux to the electron system in the interval from r_1 to r_2 due to tunneling to and from the conduction band at the interface or contact is:

$$\int_{r_1}^{r_2} H_{\mathrm{Pelt,CC}}(r)dr = \Theta[E_{\mathrm{C}}(r_1) - E_{\mathrm{C}}(r_2)]\frac{A_{\mathrm{CC}}}{qk_{\mathrm{B}}} * \int_{E_{\mathrm{C}}(r_1)}^{E_{\mathrm{C}}(r_2)} \Theta[\varepsilon - E_{\mathrm{C}}(0^-)](E_{\mathrm{C}}(r) - \varepsilon)\Gamma_{\mathrm{CC}}(r, \varepsilon) \times \tag{15.367}$$

$$\left[T_n(r)\log\left(1 + \exp\left[\frac{E_{\mathrm{F}_n}(r) - \varepsilon}{k_{\mathrm{B}}T_n(r)}\right]\right) - T_n(0^-)\log\left(1 + \exp\left[\frac{E_{\mathrm{F}_n}(0^-) - \varepsilon}{k_{\mathrm{B}}T_n(0^-)}\right]\right)\right]d\varepsilon$$

where, in the integrand of the right side, $r = r(\varepsilon)$ is the location where the energy level ε intersects the conduction band edge. The integrand of (Eq. 15.367) vanishes everywhere except at abrupt jumps of the band edge. For contacts, metals, and in presence of the `BandGap` option, the heat flux is given by (Eq. 15.367) with the Θ function removed from the integrand. The Peltier part of the heat flux to the electron system at the interface or contact due to tunneling from the conduction band in the bulk obeys an equation similar to (Eq. 15.367), but with a factor $\varepsilon - E_{\mathrm{C}}(0^-)$ rather than a factor $E_{\mathrm{C}}(r) - \varepsilon$ in the integrand, and with the integration limits $r_1 = 0^+$ and $r_2 = \infty$.

When the `Band2Band` option is used (see Table 15.123 on page 15.308), DESSIS also takes into account the band-to-band terms of the Peltier part of the heat generation. The contribution to the heat generation in the hole system due to tunneling from the conduction band at the interface or contact is:

$$\int_{r_1}^{r_2} H_{\text{Pelt,CV}}(r)dr = \Theta[E_V(r_2) - E_V(r_1)]\frac{A_{CV}}{qk_B}\int_{E_V(r_1)}^{*E_V(r_2)} \Theta[\varepsilon - E_C(0^-)](E_V(r) - \varepsilon)\Gamma_{CV}(r,\varepsilon) \times \left[T_p(r)\log\left(1 + \exp\left[\frac{E_{F_p}(r) - \varepsilon}{k_B T_p(r)}\right]\right) - T_n(0^-)\log\left(1 + \exp\left[\frac{E_{F_n}(0^-) - \varepsilon}{k_B T_n(0^-)}\right]\right)\right]d\varepsilon \tag{15.368}$$

The Peltier part of the heat flux to the electron system at the interface or contact due to tunneling from the valence band in the bulk obeys an equation similar to (Eq. 15.368), but with a factor $\varepsilon - E_C(0^-)$ rather than a factor $E_V(r) - \varepsilon$ in the integrand, and with the integration limits $r_1 = 0^+$ and $r_2 = \infty$.

The expressions for $H_{\text{Pelt,VV}}(r)$, $H_{\text{Pelt,VC}}(r)$, and the respective heat fluxes are analogous.

16.4.5.6 Implementation consideration

DESSIS computes integrals such as the one in (Eq. 15.362) as a sum of contributions from single vertices. Each vertex has a nonlocal mesh line (see Section 2.10.7 on page 15.83), and for each vertex, DESSIS performs an integration on the interval between the two intersection points of the box and the nonlocal mesh line of the vertex. To handle abrupt band edge jumps, DESSIS transforms the spatial integration to an integration over the band edge energy, for example:

$$\int_{r_1}^{r_2}[R_{CC}(r) - G_{CC}(r)]dr = \int_{E_C(r_1)}^{E_C(r_2)} \left(\frac{dE_C}{dr}(\varepsilon)\right)^{-1}(R_{CC}[r(\varepsilon)] - G_{CC}[r(\varepsilon)])d\varepsilon \tag{15.369}$$

Here, r_1 and r_2 denote the intersections of the box and nonlocal line. The derivative that appears in (Eq. 15.369) cancels with the derivatives that appear in (Eq. 15.361) and (Eq. 15.363). Hence, even for abrupt band edge jumps, the integrand of (Eq. 15.369) remains finite.

CHAPTER 17 Hot carrier injection models

17.1 Overview

Hot carrier injection is a mechanism for gate leakage. The effect is especially important for write operations in EEPROMs. DESSIS provides two hot carrier injection models:

- Classical Lucky electron injection (Maxwellian energy distribution)
- Fiegna’s hot carrier injection (non-Maxwellian energy distribution)

To activate the hot carrier injection models for electrons (holes), use the `eLucky` (`hLucky`) or `eFiegna` (`hFiegna`) options to the `GateCurrent` statement in an interface-specific `Physics` section. To activate the models for both carrier types, use the `Lucky` or `Fiegna` options. The hot carrier injection models can be combined with all tunneling models (see Chapter 16 on page 15.299). The meaning of a specification in the global `Physics` section and the `GateName` keyword are the same as for the Fowler–Nordheim model (see Section 16.2 on page 15.300).

Both hot carrier injection models are implemented as a postprocessing computation after each DESSIS simulation point. These models specify some properties of semiconductor–insulator interfaces. The most important parameter is the height of the Si–SiO$_2$ barrier (E_B). The height is a function of the insulator field F_{ins} and, at any point along the interface, it can be written as:

$$E_B = \begin{cases} E_{B0} - \alpha|F_{ins}|^{\frac{1}{2}} - \beta|F_{ins}|^{\frac{2}{3}} & F_{ins} < 0 \\ E_{B0} - \alpha|F_{ins}|^{\frac{1}{2}} - \beta|F_{ins}|^{\frac{2}{3}} + V_{ins} & F_{ins} > 0 \end{cases} \tag{15.370}$$

where E_{B0} is the zero field barrier height at the semiconductor–insulator interface. The second term in the equation represents barrier lowering due to the image potential. The third term of the barrier lowering is due to the tunneling processes. For the Si–SiO$_2$ interface, α is 2.59e-4 (Vcm)$^{1/2}$. There is a large deviation in the literature for the value of β, so it can be considered a fitting parameter.

Also, all hot carrier models contain a probability P_{ins} of scattering in the image force potential well:

$$P_{ins} = \exp\left(-\frac{x_0}{\lambda_{ins}}\right) \tag{15.371}$$

where λ_{ins} is the scattering mean free path in the insulator and the distance x_0 is given as:

$$x_0 = \sqrt{\frac{q}{16\pi\varepsilon_{ins}\varepsilon_o F_{ins}}} \tag{15.372}$$

In the above expression, ε_{ins} is the dielectric constant of the insulator.

All models below are considered only for electrons, but these expressions can also be applied to holes.

All of the hot carrier injection models below have an effective electric field F_{eff} as a parameter.

In DESSIS, there are three possibilities to calculate the effective field:

- With the electric field parallel to the carrier flow (switched on by the keyword `Eparallel`, which is default for hot carrier currents).
- With recomputation of the carrier temperature of the hydrodynamic simulation (switched on by the keyword `CarrierTempDrive`).
- With a simplified approach (compared to the second method): The drift-diffusion model is used for the device simulation, and carrier temperature is estimated as the solution of the simplified and linearized energy balance equation. As this is a postprocessing calculation, the keyword `CarrierTempPost` activates this option.

These keywords are parameters of the model keywords. For example, the Lucky electron model looks like `eLucky(CarrierTempDrive)`. The user, however, must remember that if the model includes the keyword `CarrierTempDrive`, `Hydro` and a carrier temperature calculation must be specified in the `Physics` section.

17.2 Classical Lucky electron injection

The classical total Lucky electron current from an interface to a gate contact can be written as [115]:

$$I_g = \iint J_n(x,y) P_s P_{ins} \left(\int_{E_B}^{\infty} P_\varepsilon P_r d\varepsilon \right) dx dy \qquad (15.373)$$

where $J_n(x,y)$ is the current density at any point (x,y) in the device, P_s is the probability that the electron will travel a distance y to the interface without losing any energy, P_ε is the probability that the electron has energy between ε and $\varepsilon + d\varepsilon$, P_{ins} is the probability of scattering in the image force potential well ((Eq. 15.371)), and P_r is the probability that the electron will be redirected. These probabilities are given by the expressions:

$$P_r(\varepsilon) = \frac{1}{2\lambda_r}\left(1 - \sqrt{\frac{E_{B0}}{\varepsilon}}\right) \qquad (15.374)$$

$$P_s(y) = \exp\left(-\frac{y}{\lambda}\right) \qquad (15.375)$$

$$P_\varepsilon(\varepsilon) = \frac{1}{\lambda F_{eff}} \exp\left(-\frac{\varepsilon}{\lambda F_{eff}}\right) d\varepsilon \qquad (15.376)$$

where λ is the scattering mean free path in the semiconductor, λ_r is redirection mean free path, F_{eff} is the effective electric field considered in Section 17.1 on page 15.317. E_{B0} is the height of the semiconductor–insulator barrier. The model coefficients and their defaults are given in Table 15.124 on page 15.319. They can be changed in the parameter file in the section:

```
LuckyModel { ... }
```

Table 15.124 Default coefficients for Lucky electron model

Symbol	Parameter name (Electrons)	Default value (Electrons)	Parameter name (Holes)	Default value (Holes)	Unit
λ	eLsem	8.9×10^{-7}	hLsem	1.0×10^{-7}	cm
λ_{ins}	eLins	3.2×10^{-7}	hLins	3.2×10^{-7}	cm
λ_r	eLsemR	6.2×10^{-6}	hLsemR	6.2×10^{-6}	cm
E_{B0}	eBar0	3.1	hBar0	4.7	eV
α	eBL12	2.6×10^{-4}	hBL12	2.6×10^{-4}	$(\mathrm{V} \cdot \mathrm{cm})^{1/2}$
β	eBL23	3.0×10^{-5}	hBL23	3.0×10^{-5}	$(\mathrm{V} \cdot \mathrm{cm}^2)^{1/3}$

17.3 Fiegna hot carrier injection

The total hot carrier injection current according to the Fiegna model [91] can be written as:

$$I_g = q\int P_{ins}\left(\int_{E_{B0}}^{\infty} v_{\perp}(\varepsilon)f(\varepsilon)g(\varepsilon)d\varepsilon\right)ds \tag{15.377}$$

where ε is the electron energy, E_{B0} is the height of the semiconductor–insulator barrier, $v_{\perp}$ is the velocity normal to the interface, f(ε) is the electron energy distribution, g(ε) is the density of states of the electrons, P_{ins} is the probability of scattering in the image force potential well as described by (Eq. 15.371), and $\int ds$ is an integral along the semiconductor–insulator interface.

The following expression for the electron energy distribution was proposed for a parabolic and an isotropic band structure, and equilibrium between lattice and electrons:

$$f(\varepsilon) = A\exp\left(-\chi\frac{\varepsilon^3}{F_{eff}^{1.5}}\right) \tag{15.378}$$

Therefore, the gate current can be rewritten as:

$$I_g = q\frac{A}{3\chi}\int P_{ins}n\frac{F_{\mathrm{eff}}^{3/2}}{\sqrt{E_B}}e^{-\frac{\chi E_B^3}{F_{\mathrm{eff}}^{3/2}}}ds \tag{15.379}$$

where n is the electron density and F_{eff} is an effective field considered in Section 17.1 on page 15.317. The coefficients and their defaults are given in Table 15.125.

Table 15.125 Coefficients for Fiegna model

	A [cm/s/eV$^{2.5}$]	χ [$\left(\frac{\mathrm{V}}{\mathrm{cm} \cdot \mathrm{eV}}\right)^{1.5}$]	λ_{ins} [cm]	E_{B0} [eV]	α [$(\mathrm{V} \cdot \mathrm{cm})^{1/2}$]	β [$(\mathrm{V} \cdot \mathrm{cm}^2)^{1/3}$]
Electrons	4.87×10^2	1.3×10^8	3.2×10^{-7}	3.1	2.6×10^{-4}	1.5×10^{-5}
Holes	4.87×10^2	1.3×10^8	3.2×10^{-7}	4.7	2.6×10^{-4}	1.5×10^{-5}

The above coefficients can be changed in the parameter file in the `FiegnaModel` section. Coefficients A, χ, λ_{ins}, E_{B0}, α, and β correspond to `eA`, `eChi`, `eLins`, `eBar0`, `eBL12`, and `eBL23` for electrons and `hA`, `hChi`, `hLins`, `hBar0`, `hBL12`, and `hBL23` for holes in the parameter file.

CHAPTER 18 Heterostructure device simulation

18.1 Overview

DESSIS supports the simulation of devices containing arbitrary materials and heterojunctions. In addition, it is possible to use different physical models and different parameter sets in different regions and materials of the device. This is available for both homogenous and heterojunction devices as described in Section 2.5.3 on page 15.47. To use the default material parameters and the same set of models for all regions and materials, special specifications are not required for the input file. The program automatically uses appropriate parameters for all materials defined in the geometry file.

18.2 Physics models and differential equations

Most of the models described in this manual can be applied to both homogenous semiconductors and heterostructures. Conversely, some models are applicable only for heteromaterials. In this section, the most important models are described:

- Transport equations and energy balance equations contain terms such as $\nabla\chi$, ∇E_g, and $\nabla \ln(m_e)$, which are equal to 0 for homogeneous structures (the term ∇E_g can be nonzero if the band-gap narrowing effect is taken into account), but are nonzero and crucial if a heterostructure is simulated.
- Thermionic emission models at abrupt heterointerfaces (see Section 18.10 on page 15.330).
- Tunneling at heterointerfaces (see Chapter 16 on page 15.299).

18.3 Mole fraction materials

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

1. DESSIS looks for `Molefraction.txt` in the current working directory.
2. DESSIS checks if the environment variable `DESSISDB` is defined. This variable should contain a directory or a list of directories separated by spaces or colons[1], for example:

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

   DESSIS scans the directories in the given order until `Molefraction.txt` is found.

NOTE The environment variable `DESSISDB` is also used to locate libraries of material parameters (see Section 2.13.5 on page 15.92).

1. On Windows NT, colons can be part of a directory name. Therefore, only spaces can be used to separate directories.

3. If the environment variables `ISEROOT` and `ISERELEASE` are defined, DESSIS tries to read the file:

```
$ISEROOT/tcad/$ISERELEASE/lib/dessis/MaterialDB/Molefraction.txt
```

4. If these previous strategies are unsuccessful, DESSIS uses the built-in defaults that follow.

The default `Molefraction.txt` file has the following content:

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

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

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

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

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

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

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

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

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

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

Quaternary alloys are specified by their corner materials in the file `Molefraction.txt`. For example, the 2:2 III–V quaternary alloy $In_{1-x}Ga_xAs_yP_{1-y}$ is given by:

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

and the 3:1 III–V quaternary alloy $Al_xGa_yIn_{1-x-y}As$ is defined by:

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

When the corner materials of an alloy have been specified, DESSIS determines the corresponding side materials automatically. In the case of $In_{1-x}Ga_xAs_yP_{1-y}$, the four side materials are $InAs_xP_{1-x}$, $GaAs_xP_{1-x}$, $Ga_xIn_{1-x}P$, and $In_{1-x}Ga_xAs$. Similarly, for $Al_xGa_yIn_{1-x-y}As$, there are the three side materials $In_{1-x}Al_xAs$, $Al_xGa_{1-x}As$, and $In_{1-x}Ga_xAs$.

NOTE All side and corner materials must appear in `datexcodes.txt`, and their mole dependencies must be specified in `Molefraction.txt` (see Section 2.5.4 on page 15.48).

NOTE If it is unable to parse the file `Molefraction.txt`, DESSIS reverts to the defaults shown above. This may lead to unexpected simulation results.

18.4 Mole fraction specification

In DESSIS, the mole fraction of a compound semiconductor or insulator is defined in two ways:

- In the data file (`<name>.dat`) of the device structure
- Internally, in the `Physics` section of the `des.cmd` input file

If the mole fraction is loaded from the `.dat` file and an internal mole fraction specification is also applied, the loaded mole fraction values are overwritten in the regions specified in the `MoleFraction` sections of the input file. The internal mole fraction distribution is described in the `MoleFraction` statement inside the `Physics` section:

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

The parameters for the mole fraction specification are given in Table 15.126 and grading options are described in Table 15.127 on page 15.324.

Table 15.126 Parameters for mole fraction specification

Option	Description
`xFraction=<value>`	Specifies a constant value of `xMoleFraction`.
`yFraction=<value>`	Specifies a constant value of `yMoleFraction`.
`RegionName = <reg_name>` or `RegionName = [<reg_name1> <reg_name2>...]`	Defines a region or a set of regions where the mole fraction specification will take affect.
`GrDistance=<value>:`	Specifies the distance [µm] in the direction normal to the boundaries of the specified region (or specified set of regions), where linear interpolation of mole fraction(s) from the specified constant value to 0 occurs (see below for a more sophisticated grading specification).
`Grading( (<grading_option1>) (<grading_option2>) ...)`	Another option to specify grading; allows a nonzero mole fraction and different distance of grading from different parts of the boundaries.

Table 15.127 Grading options in mole fraction specification

Option	Description
`xFraction=<value>`	Specifies the boundary `xMoleFraction` at the selected interface.
`yFraction=<value>`	Specifies the boundary `yMoleFraction` at the selected interface.
`GrDistance = <value>:`	Specifies the distance [μm] in the direction normal to the boundaries of the specified interface, where linear interpolation of mole fraction(s) from the constant value to the specified boundary mole fraction(s) occurs.
`RegionInterface = (<regionname1> <regionname2>)`	Defines an interface where this grading option is applied. Applied to all boundaries by default.

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

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

An example of a mole fraction specification is:

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

18.5 Composition-dependent models

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

- `Epsilon` (also available for compound insulators)
- `LatticeHeatCapacity` (also available for compound insulators)
- `Kappa` (lattice thermal conductivity, also available for compound insulators)

- `Bandgap`
- `eDOSMass`
- `hDOSMass`
- `ConstantMobility`
- `DopingDependence` (in the mobility models)
- `HighFieldDependence` (in the mobility models)
- `Enormal` (in the mobility models)
- `ToCurrentEnormal` (in the mobility models)
- `PhuMob` (in the mobility models)
- `vanOverstraetendeMan` (impact ionization model)
- `SchroedingerParameters`
- `AbsorptionCoefficient`
- `QWStrain` (see Section 28.7.1 on page 15.452)
- `RefractiveIndex` (also available for compound insulators)

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

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

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

18.6 Ternary semiconductor composition

To illustrate a calculation of mole fraction–dependent parameter values for ternary materials, consider one mole interval from x_{i-1} to x_i. For mole fraction value (x) of this interval, to compute the parameter value (P), DESSIS uses the expression:

$$\begin{aligned}
P &= P_{i-1} + A \cdot \Delta x + B_i \cdot \Delta x^2 + C_i \cdot \Delta x^3 \\
A &= \frac{\Delta P_i}{\Delta x_i} - B_i \cdot \Delta x_i - C_i \cdot \Delta x_i^2 \\
\Delta P_i &= P_i - P_{i-1} \\
\Delta x_i &= x_i - x_{i-1} \\
\Delta x &= x - x_{i-1}
\end{aligned} \tag{15.380}$$

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

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

$$P = (1-x)P_{x0} + xP_{x1} \quad (15.381)$$

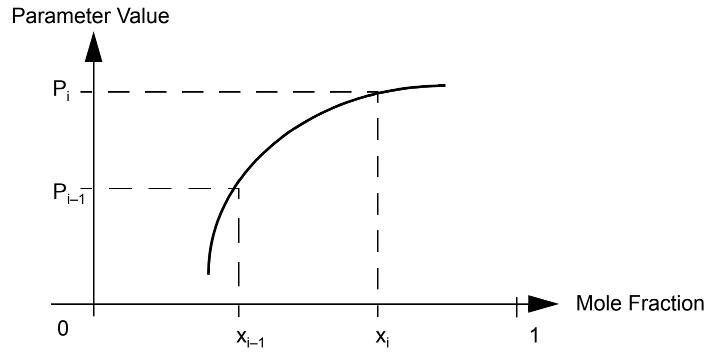


Figure 15.68 Parameter value as a function of mole fraction

Example 1: Electron effective mass specification

This example shows the specification of the parameters that define the electron effective mass, in the default section of the parameter file, for the material GaAs:

```
eDOSMass
{
  * For effective mass specification Formula1 (me approximation):
  * or Formula2 (Nc300) can be used:
Formula= 1       # [1]
  * Formula1:
  * me/m0 = [ (6 * mt)^2 *  ml ]^(1/3) + mm
  * mt = a[Eg(0)/Eg(T)]
  * Nc(T) = 2(2pi*kB/h_Planck^2*me*T)^3/2 = 2.540e19 ((me/m0)*T)^3/2
a = 0.1905          # [1]
ml = 0.9163         # [1]
mm = 0.0000e+00     # [1]
  * Formula2:
  * me/m0 = (Nc300/2.540e19)^2/3
  * Nc(T) = Nc300 * (T/300)^3/2
Nc300 = 2.8000e+19      # [cm-3]
}
```

The user can select between `Formula1` and `Formula2`. By default, one value (`Formula=...`) is chosen for each material. If necessary, the formula is also changeable in the parameter file.

For mole fraction–dependent materials, certain model parameters are specified as a function of the mole fraction. The mole fraction dependence is implemented as a piecewise polynomial approximation, up to the third order.

Example 2: Specifying dielectric permittivity

This example provides the specification of dielectric permittivity for $Al_xGa_{1-x}As$:

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

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

NOTE Although this example uses a linear interpolation over the whole interval [0,1], a higher order polynomial can be selected by specifying the appropriate parameters. Additional mole fraction intervals can also be introduced as shown in the next example.

Example 3: Specifying band gap

This example provides a specification of the band gap parameters for $Al_xGa_{1-x}As$. A polynomial approximation, up to the third degree, describes the mole fraction–dependent band parameters on every mole fraction interval. In the previous example, two intervals were used, namely, `[Xmax(0), Xmax(1)]` and `[Xmax(1), Xmax(2)]`. The parameters `Eg0`, `Chi0`, ... correspond to the values for `x=Xmax(0)`; while the parameters `Eg0(1)`, `Chi0(1)`, ... correspond to the values for `x=Xmax(1)` and, finally, `Eg0(2)`, `Chi0(2)`, ... correspond to the values for `x=Xmax(2)`. The coefficients A and F of the polynomial:

$$F + A(X - Xmin(I)) + B(X - Xmin(I))^2 + C(X - Xmin(I))^3 \tag{15.382}$$

are determined from the values at both ends of the intervals, while the coefficients B and C must be specified explicitly. The user can introduce additional intervals:

```
Bandgap *temperature dependent*
{ * Eg = Eg0 - alpha T^2 / (beta + T) + alpha Tpar^2 / (beta + Tpar)
  * Eg0 can be overwritten in below Band Gap Narrowing models,
  * if any of the BGN model is chosen in physics section.
  * Parameter 'Tpar' specifies the value of lattice
  * temperature, at which parameters below are defined.
       Eg0 = 1.42248        # [eV]
       Chi0 = 4.11826       # [eV]
       alpha = 5.4050e-04   # [eV K^-1]
       beta = 2.0400e+02    # [K]
       Tpar = 3.0000e+02    # [K]
* Mole fraction dependent model.
* The following interpolation polynomial can be used on interval [Xmin(I),Xmax(I)]:
* F(X) = F(I-1)+A(I)*(X-Xmin(I))+B(I)*(X-Xmin(I))^2+C(I)*(X-Xmin(I))^3,
* where Xmax(I), F(I), B(I), C(I) are defined below for each interval.
* A(I) is calculated for a boundary condition F(Xmax(I)) = F(I).
* Above parameters define values at the following mole fraction:
       Xmax(0) = 0.0000e+00           # [1]
* Definition of mole fraction intervals, parameters, and coefficients:
       Xmax(1) = 0.45                 # [1]
       Eg0(1) = 1.98515               # [eV]
       B(Eg0(1)) = 0.0000e+00         # [eV]
       C(Eg0(1)) = 0.0000e+00         # [eV]
```

```
            Chi0(1) = 3.575               # [eV]
            B(Chi0(1)) = 0.0000e+00       # [eV]
            C(Chi0(1)) = 0.0000e+00       # [eV]
            alpha(1) = 4.7727e-04         # [eV K^-1]
            B(alpha(1)) = 0.0000e+00      # [eV K^-1]
            C(alpha(1)) = 0.0000e+00      # [eV K^-1]
            beta(1) = 1.1220e+02          # [K]
            B(beta(1)) = 0.0000e+00       # [K]
            C(beta(1)) = 0.0000e+00       # [K]
            Xmax(2) = 1                   # [1]
            Eg0(2) = 2.23                 # [eV]
            B(Eg0(2)) = 0.143             # [eV]
            C(Eg0(2)) = 0.0000e+00        # [eV]
            Chi0(2) = 3.5                 # [eV]
            B(Chi0(2)) = 0.0000e+00       # [eV]
            C(Chi0(2)) = 0.0000e+00       # [eV]
            alpha(2) = 4.0000e-04         # [eV K^-1]
            B(alpha(2)) = 0.0000e+00      # [eV K^-1]
            C(alpha(2)) = 0.0000e+00      # [eV K^-1]
            beta(2) = 0.0000e+00          # [K]
            B(beta(2)) = 0.0000e+00       # [K]
            C(beta(2)) = 0.0000e+00       # [K]
      }
```

18.7 Quaternary semiconductor composition

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

$$AB_xC_yD_z \tag{15.383}$$

where A is a group III element, and B, C, and D are group V elements (usually listed according to increasing atomic number). Conversely, a 3:1 III-V quaternary alloy can be described as:

$$A_xB_yC_zD \tag{15.384}$$

where A, B, and C are group III elements, and D is a group V element. The composition variables x, y, and z are nonnegative, and they are constrained by:

$$x+y+z = 1 \tag{15.385}$$

An example would be $Al_xGa_yIn_{1\text{-}x\text{-}y}As$, where $1-x-y$ corresponds to z.

DESSIS uses the symmetric interpolation scheme proposed by Williams et al. [157] to compute the parameter value $P(A_xB_yC_zD)$ of a 3:1 III–V quaternary alloy as a weighted sum of the corresponding ternary values:

$$P(A_xB_yC_zD) = \frac{xyP(A_{1-u}B_uD)+yzP(B_{1-v}C_vD)+xzP(A_{1-w}C_wD)}{xy+yz+xz} \tag{15.386}$$

where:

$$u = \frac{1-x+y}{2}, v = \frac{1-y+z}{2}, w = \frac{1-x+z}{2} \tag{15.387}$$

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

$$A_xB_{1-x}C_yD_{1-y} \tag{15.388}$$

where A and B are group III elements, and C and D are group V elements. The composition variables x and y satisfy the inequalities $0 \le x \le 1$ and $0 \le y \le 1$. As an example, we mention the material $In_{1\text{-}x}Ga_xAs_yP_{1\text{-}y}$.

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

$$P(A_xB_{1-x}C_yD_{1-y}) = \frac{x(1-x)(yP(A_xB_{1-x}C)+(1-y)P(A_xB_{1-x}D))+y(1-y)(xP(AC_yD_{1-y})+(1-x)P(BC_yD_{1-y}))}{x(1-x)+y(1-y)} \tag{15.389}$$

The interpolation of model parameters for quaternary alloys is also discussed in the literature [159]–[162]. A comprehensive survey paper is available [158].

18.8 Default model parameters for compound semiconductors

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

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

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

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

Sometimes, it is difficult to analyze such a parameter file to obtain a real value of physical models (for example, the band gap) for certain composition mole fraction. By using the command `dessis -M <inputfile.cmd>`, DESSIS creates a `dessis-M.par` file that will contain regionwise parameters with only constant values (instead of the polynomial coefficients) for regions where the composition mole fraction is constant. For regions where the composition is not a constant, DESSIS prints the default material parameters.

18.9 Abrupt and graded heterojunctions

DESSIS is designed to support both abrupt and graded heterojunctions, with an arbitrary mole fraction distribution. As previously described, a piecewise polynomial interpolation is used for interpolation of parameters of the physical models over the mole fraction *x*. In the case of abrupt heterojunctions, DESSIS treats discontinuous datasets properly by introducing double points at the heterointerfaces.

This option is switched on automatically when thermionic emission (see Section 18.10) or tunneling models (see Chapter 16 on page 15.299) are selected, or when the keyword `HeteroInterface` is specified in the `Physics` section of a selected heterointerface. By default, this double points option is switched off.

NOTE The keyword `HeteroInterface` provides the equilibrium conditions for the double points. It gives quasi-Fermi potential and a simple, exponential relation between concentrations, but it does not follow real physics at the interface for high current regimes. The `HeteroInterface` option without the thermionic emission and tunneling models can be used for experimental simulations only.

To illustrate the double points option, Figure 15.69 shows a distribution of the conduction band near an abrupt heterointerface. The wide line shows a case without double points, which requires a very fine mesh to avoid a large barrier error (δE_C).

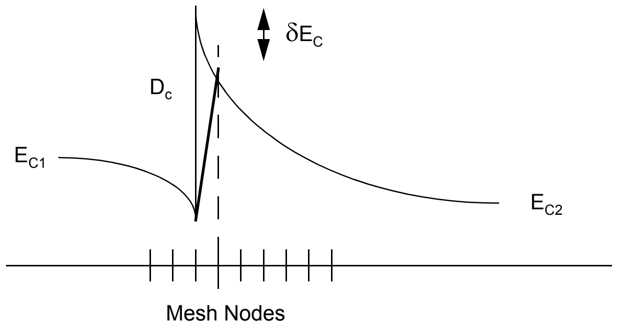


Figure 15.69 Heterostructure barrier representation with and without double points

18.10 Thermionic emission current

Conventional transport equations cease to be valid at a heterojunction interface, and currents and energy fluxes at the abrupt interface between two materials are better defined by the interface condition at the heterojunction. In defining thermionic current and thermionic energy flux, DESSIS follows the literature [124].

18.10.1 Syntax and implementation

To activate the thermionic current model for electrons at a region-interface (material-interface) heterojunction, the keyword `eThermionic` must be specified in the appropriate region-interface (material-interface) `Physics` section, for example:

```
Physics(MaterialInterface="GaAs/AlGaAs") {
   eThermionic }
```

Similarly, to activate thermionic current for holes, the keyword `hThermionic` must be specified. The keyword `Thermionic` activates the thermionic emission model for both electrons and holes. If any of these keywords is specified in the `Physics` section for a region `Region.0`, where `Region.0` is a semiconductor, the appropriate model will be applied to each `Region.0`–semiconductor interface. For small particle and energy fluxes across the interface, the condition of continuous quasi-Fermi level and carrier temperature is sometimes used. This option is activated by the keyword `Heterointerface` in the appropriate `Physics` section.

NOTE In realistic transistors, such an approach may lead to unsatisfactory results [125].

The user can change the default values of the coefficients of the thermionic emission model in the `ThermionicEmission` section of the parameter file:

```
ThermionicEmission {
   A = 2, 2   # [1]
   B = 4, 4   # [1]
   C = 1, 1   # [1]
}
```

18.10.2 Model description

Assume that at the heterointerface between materials 1 and 2, the conduction edge jump is positive, that is $\Delta E_C > 0$, where $\Delta E_C = E_{C2} - E_{C1}$ (that is, $\chi_1 > \chi_2$). If $J_{n,2}$ and $S_{n,2}$ are the electron current density and electron energy flux density entering material 2, and $J_{n,1}$ and $S_{n,1}$ are the electron current density and electron energy flux density leaving material 1, the interface condition can be written as:

$$J_{n,2} = J_{n,1} \tag{15.390}$$

$$J_{n,2} = aq\left[v_{n,2}n_2 - \frac{m_2}{m_1}v_{n,1}n_1 \exp\left(-\frac{\Delta E_C}{k_B T_{e,1}}\right)\right] \tag{15.391}$$

$$S_{n,2} = S_{n,1} + \frac{c}{q}J_{n,2}\Delta E_C \tag{15.392}$$

$$S_{n,2} = (-b)\left[v_{n,2}n_2 k_B T_{e,2} - \frac{m_2}{m_1}v_{n,1}n_1 k_B T_{e,1}\exp\left(-\frac{\Delta E_C}{k_B T_{e,1}}\right)\right] \tag{15.393}$$

where the ‘emission velocities’ are defined as:

$$v_{n,i} = \sqrt{\frac{k_B T_{e,i}}{2\pi m_i}} \tag{15.394}$$

and by default, the coefficients in the above equations are *a*=2, *b*=4, and *c*=1, which corresponds to the literature [124]. Similar equations for the hole thermionic current and hole thermionic energy flux are presented below:

$$J_{p,2} = J_{p,1} \tag{15.395}$$

$$J_{p,2} = (-a_h)q\left[v_{p,2}p_2 - \frac{m_2}{m_1}v_{p,1}p_1 \exp\left(\frac{\Delta E_V}{k_B T_{h,1}}\right)\right] \tag{15.396}$$

$$S_{p,2} = S_{p,1} + \frac{c_h}{q} J_{p,2} \Delta E_V \tag{15.397}$$

$$S_{p,2} = (-b_h)\left[v_{p,2} p_2 k_B T_{h,2} - \frac{m_2}{m_1} v_{p,1} n_1 k_B T_{h,1} \exp\left(\frac{\Delta E_V}{k_B T_{h,1}}\right)\right] \tag{15.398}$$

$$v_{p,i} = \sqrt{\frac{k_B T_{h,i}}{2\pi m_i}} \tag{15.399}$$

An equivalent set of equations are used if Fermi carrier statistics are selected.

CHAPTER 19 Energy-dependent parameters

19.1 Overview

DESSIS provides the possibility to specify some parameters as a ratio of two irrational polynomials. The general form of such ratio is written as:

$$G(w, s) = f\frac{((\sum a_i w^{p_i}) + d_n s)^{g_n}}{((\sum a_j w^{p_j}) + d_d s)^{g_d}} \qquad (15.400)$$

where subscripts n and d corresponds to numerator and denominator, respectively; *f* is a factor, *w* is a primary variable, and *s* is an additional variable. It is possible to use (Eq. 15.400) with different coefficients for different intervals k defined by the segment $[w_{k-1}^{max}, w_k^{max}]$. By default, it is assumed that only one interval k=0 with the boundaries $[0,\infty]$ exists, and function G is constant, that is, $a_0 = 0$, $a_i = 0$, $p = d = 0$, $g = 1$. Factor f is defined accordingly for each model.

A simplified syntax is introduced to define the piecewise linear function G. The boundaries of the intervals and the value of factor must be specified, which means the value of G is at the right side of the interval. All other coefficients should not be specified to use this possibility. As there are some peculiarities in parameter specification and model activation, the specific models for which the approximation by (Eq. 15.400) is supported are described here separately.

19.2 Energy-dependent energy relaxation time

For the specification of the energy relaxation time, the following modification of (Eq. 15.400) is used:

$$\tau(w) = \tau_w^0 \frac{((\sum a_i w^{p_i}))^{g_n}}{((\sum a_j w^{p_j}))^{g_d}} \qquad (15.401)$$

where $w = 1.5(kT_n)/q$ for electrons and $w = (kT_p)/q$ for holes. The factor *f* in (Eq. 15.400) is defined by τ_w^0, which can be specified in the parameter file by the values `(tau_w)_ele` and `(tau_w)_hol`.

To activate the specification of the energy-dependent energy relaxation time, the parameter `Formula(tau_w)_ele` (or `Formula(tau_w)_hol` for holes) must be set to 2. The following example shows the energy relaxation time section of the parameter file and provides a short description of the syntax:

```
EnergyRelaxationTime
{ *  Energy relaxation times in picoseconds
          (tau_w)_ele = 0.3    # [ps]
          (tau_w)_hol = 0.25   # [ps]
 * Below is the example of energy relaxation time approximation
 * by the ratio of two irrational polynomials.
 * If Wmax(interval-1) < Wc < Wmax(interval), then:
 * tau_w = (tau_w)*(Numerator^Gn)/(Denominator^Gd),
 * where (Numerator or Denominator)=SIGMA[A(i)(Wc^P(i))],
 * Wc=1.5(k*Tcar)/q (in eV).
```

```
      * By default: Wmin(0)=Wmax(-1)=0; Wmax(0)=infinity.
      * The option can be activated by specifying appropriate Formula equals 2
      *      Formula(tau_w)_ele = 2
      *      Formula(tau_w)_hol = 2
      *      Wmax(interval)_ele =
      *      (tau_w)_ele(interval) =
      *      Numerator(interval)_ele{
      *        A(0)  =
      *        P(0)  =
      *        A(1)  =
      *        P(1)  =
      *        G     =
      *      }
      *      Denominator(interval)_ele{
      *        A(0)  =
      *        P(0)  =
      *        G     =
      *      }
      *      Wmax(interval)_hol =
      *      (tau_w)_hol(interval) =
            (tau_w)_ele = 0.3       # [ps]
            (tau_w)_hol = 0.25      # [ps]

            Formula(tau_w)_ele = 2
            Numerator(0)_ele{
              A(0)  = 0.048200
              P(0)  = 0.00
              A(1)  = 1.00
              P(1)  = 3.500
              A(2)  = 0.0500
              P(2)  = 2.500
              A(3)  = 0.0018100
              P(3)  = 1.00
            }
            Denominator(0)_ele{
              A(0)  = 0.048200
              P(0)  = 0.00
              A(1)  = 1.00
              P(1)  = 3.500
              A(2)  = 0.100
              P(2)  = 2.500
            }
```

The following example shows a simplified syntax for piecewise linear specification of energy relaxation time:

```
EnergyRelaxationTime:
{ *  Energy relaxation times in picoseconds
          (tau_w)_ele = 0.3           # [ps]
          (tau_w)_hol = 0.25          # [ps]
       Formula(tau_w)_ele = 2
       Wmax(0)_ele = 0.5
       (tau_w)_ele(1) = 0.46          # [ps]
       Wmax(1)_ele = 1.
       (tau_w)_ele(2) = 0.4           # [ps]
       Wmax(2)_ele = 2.
       (tau_w)_ele(3) = 0.2           # [ps]
}
```

DESSIS also allows spline approximation of energy relaxation time over energy. In this case, the parameter `Formula(tau_w)_ele` for electron energy relaxation time (and similarly, parameter `Formula(tau_w)_hol` for hole energy relaxation time) must be equal to 3.

Inside the braces following the keyword `Spline(tau_w)_ele` (or `Spline(tau_w)_hol`), an energy [eV] and tau [ps] value pair must be specified in each line. For the values outside of the specified intervals, energy relaxation time is treated as a constant and equal to the closest boundary value.

The following example shows a spline approximation specification for energy-dependent energy relaxation time for electrons:

```
EnergyRelaxationTime {
   Formula(tau_w)_ele = 3
   Spline(tau_w)_ele {
      0.     0.3
      0.5.   0.46
      1.     0.4
      2.     0.2
   }
}
```

19.3 Energy-dependent mobility

In addition to the existing energy-dependent mobility models (such as Caughey–Thomas, where the effective field is computed inside DESSIS as a function of the carrier temperature), a more complex, user-supplied mobility model can be defined. For such specification of energy-dependent mobility, a modification to (Eq. 15.400) is used:

$$\mu(\bar{w}, N_i) = \mu_{low}\frac{\left(\left(\sum a_i \bar{w}^{p_i}\right) + d_n N_i\right)^{g_n}}{\left(\left(\sum a_j \bar{w}^{p_j}\right) + d_d N_i\right)^{g_d}} \qquad (15.402)$$

where $\bar{w} = T_n/T_L$ for electrons or $\bar{w} = T_p/T_L$ for holes, μ_{low} is a low field mobility, and N_i is a total doping concentration.

To activate the model, the driving force keyword `CarrierTemperatureDrivePolynomial` must be specified as a parameter of the high-field saturation mobility model. Parameters of the polynomials must be defined in the `HydroHighFieldMobility` section of the parameter file.

This example shows the output of the `HydroHighFieldMobility` section and the specification of coefficients:

```
HydroHighFieldDependence:
{ * Parameter specifications for the high field degradation in
   * some hydrodynamic models.
   * B) Approximation by the ratio of two irrational polynomials
  * (driving force 'CarrierTempDrivePolynomial'):
  * If Wmax(interval-1) < w < Wmax(interval), then:
  * mu_hf = mu*factor*(Numerator^Gn)/(Denominator^Gd),
  * where (Numerator or Denominator)={SIGMA[A(i)(w^P(i))]+D*Ni},
  * w=Tc/Tl; Ni(cm^-3) is total doping.
  * By default: Wmin(0)=Wmax(-1)=0; Wmax(0)=infinity.

  *      Wmax(interval)_ele =
  *      F(interval)_ele =
  *      Numerator(interval)_ele{
  *        A(0)  =
  *        P(0)  =
  *        A(1)  =
  *        P(1)  =
  *        D     =
*        G     =
*      }
*      Denominator(interval)_ele{
*        A(0)  =
*        P(0)  =
*        D     =
*        G     =
*      }
*      F(interval)_hol =
*      Wmax(interval)_hol =
      Denominator(0)_ele
{
         A(0)  = 0.3
         P(0)  = 0.0
         A(1)  = 1.0
         P(1)  = 2.
         A(2)  = 0.001
         P(2)  = 2.500
         D     = 3.00e-16
         G     = 0.2500
      }
}
```

19.4 Energy-dependent Peltier coefficient

DESSIS allows for the following modification of the expression of the energy flux equation:

$$\vec{S}_n = -\frac{5r_n}{2}\left(\frac{k_B T_n}{q}\vec{J}_n + f_n^{hf}\hat{\kappa}_n \frac{\partial(w_n \Pi_n)}{\partial w_n}\nabla T_n\right) \tag{15.403}$$

The standard expression corresponds to $\Pi_n = 1$. If $\Pi_n = 1 + P(\overline{w})$, then:

$$\frac{\partial(w_n \Pi_n)}{\partial w_n} = 1 + \overline{w}\,\frac{\partial P(\overline{w})C}{\partial \overline{w}} \tag{15.404}$$

DESSIS allows the user to specify the function Q:

$$Q(\overline{w}) = \overline{w}\,\frac{\partial P(\overline{w})}{\partial \overline{w}} \tag{15.405}$$

For the specification of Q, the following modification of (Eq. 15.400) is used:

$$Q(\overline{w}) = f\frac{\left(\left(\sum a_i \overline{w}^{p_i}\right)\right)^{g_n}}{\left(\left(\sum a_j \overline{w}^{p_j}\right)\right)^{g_d}} \tag{15.406}$$

Coefficients must be specified in the `HeatFlux` section of the parameter file, and the dependence can be activated by specifying a nonzero factor f.

For $\Pi_n = 1 + 1/(\sqrt{1+\overline{w}^2})$, the result is $Q = 1/(\overline{w}^2+1)^{1.5}$. This is an example of the parameter file section for such a function Q_n:

```
HeatFlux
{ *  Heat flux factor (0 <= hf <= 1)
       hf_n = 1      # [1]
       hf_p = 1      # [1]
 * Coefficients can be defined also as:
 *      hf_new = hf*(1.+Delta(w))
 * where Delta(w) is the ratio of two irrational polynomials.
 * If Wmax(interval-1) < Wc < Wmax(interval), then:
 * Delta(w) = factor*(Numerator^Gn)/(Denominator^Gd),
 * where (Numerator or Denominator)=SIGMA[A(i)(w^P(i))], w=Tc/Tl
 * By default: Wmin(0)=Wmax(-1)=0; Wmax(0)=infinity.
 * Option can be activated by specifying nonzero 'factor'.
 *      Wmax(interval)_ele =
 *      F(interval)_ele = 1
 *      Numerator(interval)_ele{
 *        A(0)  =
 *        P(0)  =
 *        A(1)  =
 *        P(1)  =
 *        G     =
 *      }
 *      Denominator(interval)_ele{
 *        A(0)  =
 *        P(0)  =
 *        G     =
 *      }
 *      Wmax(interval)_hol =
 *      F(interval)_hol = 1
        f(0)_ele = 1
       Denominator(0)_ele{
         A(0)  = 1.
         P(0)  = 0.
         A(1)  = 1.
         P(1)  = 2.
         G     = 1.5
       }
```

CHAPTER 20 Anisotropic properties

DESSIS allows for the modeling of the anisotropic properties of certain semiconductors.

20.1 Anisotropic mobility

In some semiconductors, such as silicon carbide, the electrons and holes may exhibit different mobilities along different crystallographic axes.

20.1.1 Crystal reference system

The crystal reference system of the semiconductor can be specified in the DESSIS parameter file as follows:

```
LatticeParameters {
   X = (1, 0, 0)
   Y = (0, 1, 0)
}
```

Instead of `LatticeParameters`, the keywords `Piezo` or `PiezoParameters` are recognized as well.

By default, DESSIS uses `X=(1,0,0)`, `Y=(0,1,0)`, and `Z=(0,0,1)`.

20.1.2 Anisotropy factor

In a 3D simulation, DESSIS assumes that the electrons or holes exhibit a mobility μ along the x and y axes, and an anisotropic mobility μ_{aniso} along z. In a 2D simulation, the regular mobility μ is observed along the x-axis, and μ_{aniso} is observed along the y-axis. The anisotropy factor r is defined as the ratio:

$$r = \frac{\mu}{\mu_{\text{aniso}}} \tag{15.407}$$

20.1.3 Current densities

In the isotropic case, the current densities can be expressed by:

$$\vec{J}_n = \mu_n \vec{g}_n \tag{15.408}$$

$$\vec{J}_p = \mu_p \vec{g}_p \tag{15.409}$$

where $\vec{g}_n$ and $\vec{g}_p$ are the *currents without mobilities*. In the drift-diffusion model, we have:

$$\vec{g}_n = -nq\nabla\phi_n \tag{15.410}$$

$$\vec{g}_p = -pq\nabla\phi_p \tag{15.411}$$

as can be seen from (Eq. 15.21) and (Eq. 15.22). For the thermodynamic model, (Eq. 15.23) and (Eq. 15.24) imply that:

$$\vec{g}_n = -nq(\nabla\phi_n + P_n\nabla T) \tag{15.412}$$

$$\vec{g}_p = -pq(\nabla\phi_p + P_p\nabla T) \tag{15.413}$$

In the hydrodynamic case, we have:

$$\vec{g}_n = n\nabla E_C + k_B T_n \nabla n + f_n^{td} k_B n \nabla T_n - 1.5 n k_B T_n \nabla \ln m_e \tag{15.414}$$

$$\vec{g}_p = p\nabla E_V - k_B T_p \nabla p - f_p^{td} k_B p \nabla T_p - 1.5 p k_B T_p \nabla \ln m_h \tag{15.415}$$

according to (Eq. 15.26) and (Eq. 15.27). For anisotropic mobilities, (Eq. 15.408) and (Eq. 15.409) need to be rewritten as:

$$\vec{J}_n = \mu_n A_{r_n} \vec{g}_n \tag{15.416}$$

$$\vec{J}_p = \mu_p A_{r_p} \vec{g}_p \tag{15.417}$$

where r_n and r_p are the anisotropy factors for electrons and holes, respectively. If the crystal reference system coincides with the DESSIS coordinate system, the matrices A_r are given by:

$$A_r = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1/r \end{bmatrix} \text{ or } \mathrm{A}_r = \begin{bmatrix} 1 & \\ & 1/r \end{bmatrix} \tag{15.418}$$

depending on the dimension of the problem. In general, however, A_r needs to be written as:

$$A_r = Q\begin{bmatrix} 1 & & \\ & 1 & \\ & & 1/r \end{bmatrix}Q^T \text{ or } \mathrm{A}_r = Q_{2:2}\begin{bmatrix} 1 & \\ & 1/r \end{bmatrix}Q_{2:2}^T \tag{15.419}$$

where Q is the 3-by-3 orthogonal matrix:

$$Q = \begin{bmatrix} \dfrac{\vec{\mathrm{X}}}{\|\vec{\mathrm{X}}\|_2} & \dfrac{\vec{\mathrm{Y}}}{\|\vec{\mathrm{Y}}\|_2} & \dfrac{\vec{\mathrm{Z}}}{\|\vec{\mathrm{Z}}\|_2} \end{bmatrix} \tag{15.420}$$

and the quantity $Q_{2:2}$ denotes the leading 2-by-2 submatrix of Q. For 2D problems, the vectors $\vec{\mathrm{X}}$ and $\vec{\mathrm{Y}}$ must lie in the x-y plane.

Therefore, Q will have the form:

$$Q = \begin{bmatrix} x & x & 0 \\ x & x & 0 \\ 0 & 0 & \pm 1 \end{bmatrix} = \begin{bmatrix} Q_{2:2} & 0 \\ 0 & \pm 1 \end{bmatrix} \tag{15.421}$$

20.1.4 Driving forces

In the isotropic case, the electric field parallel to the electron or hole current is given by:

$$F_c = \frac{(\vec{E}, \vec{J}_c)}{\sqrt{(\vec{J}_c, \vec{J}_c)}} = \frac{(\vec{E}, \vec{g}_c)}{\sqrt{(\vec{g}_c, \vec{g}_c)}} \tag{15.422}$$

where $(.,.)$ denotes the inner product between two vectors (see (Eq. 15.190)). For anisotropic mobilities:

$$F_c = \frac{(\vec{E}, A\vec{g}_c)}{\sqrt{(A\vec{g}_c, A\vec{g}_c)}} \tag{15.423}$$

Similarly, the electric field perpendicular to the current, as given in (Eq. 15.166), needs to be rewritten as:

$$F_{c,\perp} = \sqrt{(\vec{E}, \vec{E})^2 - \frac{(\vec{E}, A\vec{g}_c)^2}{(A\vec{g}_c, A\vec{g}_c)}} \tag{15.424}$$

(Eq. 15.191) shows how the gradient of the Fermi potential φ_c may be used as the driving force in high-field saturation models. Instead, for anisotropic mobilities, DESSIS uses:

$$F_c = A\nabla\varphi_c \tag{15.425}$$

In the isotropic hydrodynamic Canali model, the driving force $\vec{F}_c$ satisfies:

$$(\mu\vec{F}_c, \vec{F}_c) = \frac{w_c - w_0}{\tau_{e,c} q} \tag{15.426}$$

as can be seen from (Eq. 15.192). To derive the appropriate expression in the anisotropic case we assume that $\vec{F}_c$ operates parallel to the current, that is:

$$\vec{F}_c = F_c \vec{e}_c \tag{15.427}$$

where:

$$\vec{e}_c = \frac{A\vec{g}_c}{\left\| A\vec{g}_c \right\|} \tag{15.428}$$

is the direction of the electron or hole current.

Instead of (Eq. 15.426), we now have:

$$\mu F_c^2(A\vec{e}_c,\vec{e}_c) = \frac{w_c - w_0}{\tau_{e,c} q} \tag{15.429}$$

or:

$$F_c = \sqrt{\frac{w_c - w_0}{\tau_{e,c} q \mu(A\vec{e}_c,\vec{e}_c)}} \tag{15.430}$$

20.1.5 Total anisotropic mobility

This is the simplest mode in DESSIS. Only a total anisotropy factor r_e or r_h is specified in the command file:

```
Physics {
   Aniso(
      eMobilityFactor (Total) = re
      hMobilityFactor (Total) = rh
   )
}
```

DESSIS computes the mobility μ for electrons or holes along the main crystallographic axis as usual. The mobility μ_{aniso} is then given by:

$$\mu_{\text{aniso}} = \frac{\mu}{r} \tag{15.431}$$

NOTE In this mode, DESSIS does not update the driving forces as discussed in Section 20.1.4 on page 15.341.

20.1.6 Total direction-dependent anisotropic mobility

This mode is activated by specifying the total direction-dependent anisotropy factor r_e or r_h in the DESSIS command file:

```
Physics {
   Aniso(
      eMobilityFactor (TotalDD) = re
      hMobilityFactor (TotalDD) = rh
   )
}
```

First, DESSIS updates the driving forces as discussed in Section 20.1.4. Afterwards, the electron or hole mobility μ along the main crystallographic axis is computed as usual, and the mobility μ_{aniso} is given by:

$$\mu_{\text{aniso}} = \frac{\mu}{r} \tag{15.432}$$

20.1.7 Self-consistent anisotropic mobility

This is the most accurate, but also the most expensive, mode in DESSIS. The electron and hole mobility models specified in the `Physics` section are evaluated separately for the major and minor crystallographic axes, but with different parameters for each axis. This option is activated in the `Physics` section for electron or hole mobilities as follows:

```
Physics {
   Aniso(
      eMobility
      hMobility
   )
}
```

To simplify matters, it is possible to specify:

```
Physics {
   Aniso(
      Mobility
   )
}
```

to activate self-consistent, anisotropic, mobility calculations for both electrons and holes. Table 15.128 lists the mobility models that offer an anisotropic version.

Table 15.128 Anisotropic mobility models

Isotropic model	**Anisotrophic model**
`ConstantMobility`	`ConstantMobility_aniso`
`DopingDependence`	`DopingDependence_aniso`
`EnormalDependence`	`EnormalDependence_aniso`
`HighFieldDependence`	`HighFieldDependence_aniso`
`UniBoDopingDependence`	`UniBoDopingDependence_aniso`
`UniBoEnormalDependence`	`UniBoEnormalDependence_aniso`
`UniBoHighFieldDependence`	`UniBoHighFieldDependence_aniso`
`HydroHighFieldDependence`	`HydroHighFieldDependence_aniso`

The PMI also supports anisotropic mobility calculations. The constructors of the classes `PMI_DopingDepMobility`, `PMI_EnormalMobility`, and `PMI_HighFieldMobility` contain an additional flag to distinguish between the isotropic and anisotropic case (see Chapter 33 on page 15.535). For example, users can specify these parameters for the constant mobility model in the DESSIS parameter file:

```
ConstantMobility {
   mumax = 1.4170e+03, 4.7050e+02
   Exponent = 2.5, 2.2
}
```

The following parameters would then compute a reduced constant mobility along the anisotropic axis:

```
ConstantMobility_aniso {
   mumax = 1.0e+03, 4.0e+02
   Exponent = 2.5, 2.2
}
```

In each vertex, DESSIS introduces the anisotropy factors r_e and r_h as two additional unknowns. For a given value of r, the driving forces F_c and $F_{c,\perp}$ are computed as discussed in Section 20.1.4 on page 15.341, and the mobilities along the isotropic and anisotropic axes are obtained.

The equation for the unknown factor r is then given by:

$$r = \frac{\mu(r)}{\mu_{\text{aniso}}(r)} \tag{15.433}$$

This nonlinear equation is solved in each vertex for both electron and hole mobilities.

20.1.8 Math section

Table 15.129 lists the options that pertain to anisotropic mobility calculations.

Table 15.129 Math options for anisotropic mobility calculations

Option	Description
`NewAniso`	This flag activates the new implementation of anisotropic mobilities. It is active by default and is switched off by specifying `-NewAniso`.
`TensorGridAniso`	The anisotropic effects are modeled using a tensor grid approximation. This option can only be used for total anisotropic mobility on tensor grids or near-tensor grids.

20.1.9 Plot section

Table 15.130 lists the plot variables that may be useful for visualizing anisotropic mobility calculations.

Table 15.130 Plot variables for anisotropic mobility

Plot variable	Description
`eMobility`	Electron mobility along main axis
`hMobility`	Hole mobility along main axis
`eMobilityAniso`	Electron mobility along anisotropic axis
`hMobilityAniso`	Hole mobility along anisotropic axis
`eMobilityAnisoFactor`	Anisotropic factor for electrons
`hMobilityAnisoFactor`	Anisotropic factor for holes

20.2 Anisotropic avalanche generation

DESSIS computes the avalanche generation according to (Eq. 15.238). In the isotropic case, the terms nv_n and pv_p can also be written as:

$$nv_n = \mu_n \left\| \vec{g}_n \right\| \tag{15.434}$$

and:

$$pv_p = \mu_p \|\vec{g}_p\| \tag{15.435}$$

respectively. If anisotropic mobilities are switched on, (Eq. 15.434) and (Eq. 15.435) are replaced by:

$$nv_n = \mu_n \|A_{r_n} \vec{g}_n\| \tag{15.436}$$

and:

$$pv_p = \mu_p \|A_{r_p} \vec{g}_p\| \tag{15.437}$$

NOTE (Eq. 15.436) and (Eq. 15.437) only apply to total direction–dependent and self-consistent mobility calculations. If the total anisotropic option (see Section 20.1.5 on page 15.342) is selected, (Eq. 15.434) and (Eq. 15.435) are used.

Anisotropic avalanche calculations can be activated in the `Physics` section, independently for electrons and holes:

```
Physics {
   Aniso(
      eAvalanche
      hAvalanche
   )
}
```

The keyword `Avalanche` activates calculations of anisotropic avalanche for both electrons and holes:

```
Physics {
   Aniso(
      Avalanche
   )
}
```

In the anisotropic mode, different avalanche parameters can be specified along the isotropic and anisotropic axes. Table 15.131 shows the avalanche models that are supported.

Table 15.131 Anisotropic avalanche models

Isotropic model	**Anisotrophic model**
`vanOverstraetendeMan`	`vanOverstraetendeMan_aniso`
`OkutoCrowell`	`OkutoCrowell_aniso`

DESSIS uses interpolation to compute avalanche parameters for an arbitrary direction of the current. Let $\vec{e}_c$ be the direction of the electron or hole current as defined in (Eq. 15.428). In the crystal reference system, the current is given by:

$$\vec{\varepsilon}_c = Q^T \vec{e}_c = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \end{bmatrix} \tag{15.438}$$

NOTE Both vectors $\vec{e}_c$ and $\vec{\varepsilon}_c$ are of unit length.

DESSIS interpolates an avalanche parameter p depending on the direction of the current $\vec{e}_c$ according to:

$$p(\vec{e}_c) = (\varepsilon_x^2 + \varepsilon_y^2) \cdot p_{\text{isotropic}} + \varepsilon_z^2 \cdot p_{\text{anisotropic}} \tag{15.439}$$

in a 3D simulation, and, in a 2D simulation:

$$p(\vec{e}_c) = \varepsilon_x^2 \cdot p_{\text{isotropic}} + \varepsilon_y^2 \cdot p_{\text{anisotropic}} \tag{15.440}$$

The PMI also supports the calculation of anisotropic avalanche generation. The current without mobility $A\vec{g}_c$ is passed as an input parameter, and it can be used by the PMI code to determine the model parameters depending on the direction of the current (see Section 33.8 on page 15.544).

20.3 Anisotropic electrical permittivity

The electrical permittivity ε in (Eq. 15.19) can have different values along different crystallographic axes. If the crystallographic axes coincide with the DESSIS coordinate system, the scalar ε is replaced by the matrix:

$$E = \begin{bmatrix} \varepsilon & & \\ & \varepsilon & \\ & & \varepsilon_{\text{aniso}} \end{bmatrix} \text{ or } E = \begin{bmatrix} \varepsilon & \\ & \varepsilon_{\text{aniso}} \end{bmatrix} \tag{15.441}$$

depending on the dimension of the problem. For general crystallographic axes, the matrix E is given by:

$$E = Q \begin{bmatrix} \varepsilon & & \\ & \varepsilon & \\ & & \varepsilon_{\text{aniso}} \end{bmatrix} Q^T \text{ or } E = Q_{2:2} \begin{bmatrix} \varepsilon & \\ & \varepsilon_{\text{aniso}} \end{bmatrix} Q_{2:2}^T \tag{15.442}$$

where Q is defined in (Eq. 15.420).

Anisotropic electrical permittivity is switched on by using the keyword `Poisson` in the `Physics` section of the DESSIS command file:

```
Physics {
   Aniso (Poisson)
}
```

This command should only appear in the global `Physics` section. Regionwise activation of anisotropic electrical permittivity is not supported.

The model parameters for ε and $\varepsilon_{\text{aniso}}$ can be specified in the DESSIS parameter file. Table 15.132 lists the names of the corresponding models.

Table 15.132 Anisotropic electrical permittivity models

Isotropic model	**Anisotrophic model**
`Epsilon`	`Epsilon_aniso`

Different parameters can be specified for each region or each material. The following statement in the DESSIS command file can be used to plot the electrical permittivities:

```
Plot {
   DielectricConstant
   "DielectricConstantAniso"
}
```

20.4 Anisotropic thermal conductivity

The thermal conductivity κ in (Eq. 15.25) can have different values along different crystallographic axes. If the crystallographic axes coincide with the DESSIS coordinate system, the scalar κ is replaced by the matrix:

$$K = \begin{bmatrix} \kappa & & \\ & \kappa & \\ & & \kappa_{\text{aniso}} \end{bmatrix} \text{ or } K = \begin{bmatrix} \kappa & \\ & \kappa_{\text{aniso}} \end{bmatrix} \tag{15.443}$$

depending on the dimension of the problem. For general crystallographic axes, the matrix K is given by:

$$K = Q\begin{bmatrix} \kappa & & \\ & \kappa & \\ & & \kappa_{\text{aniso}} \end{bmatrix}Q^T \text{ or } K = Q_{2:2}\begin{bmatrix} \kappa & \\ & \kappa_{\text{aniso}} \end{bmatrix}Q_{2:2}^T \tag{15.444}$$

where Q is defined in (Eq. 15.420).

Anisotropic thermal conductivity is switched on by the keyword `Temperature` in the `Physics` section of the DESSIS command file:

```
Physics {
   Aniso (Temperature)
}
```

This command should only appear in the global `Physics` section. Regionwise activation of anisotropic thermal conductivity is not supported.

The model parameters for κ and κ_{aniso} can be specified in the DESSIS parameter file. Table 15.133 lists the names of the corresponding models.

Table 15.133 Anisotropic thermal conductivity models

Isotropic model	Anisotrophic model
Kappa	Kappa_aniso

Different parameters can be specified for each region or each material. The PMI can also be used to compute anisotropic thermal conductivities. The constructor of the class `PMI_ThermalConductivity` has an additional parameter to distinguish between the isotropic and anisotropic directions (see Section 33.20 on page 15.581). The following statement in the DESSIS command file can be used to plot the thermal conductivities:

```
Plot {
   "ThermalConductivity"
      "ThermalConductivityAniso"
}
```

CHAPTER 21 Ferroelectric materials

21.1 Overview

In ferroelectric materials, the polarization $\vec{P}$ depends nonlinearly on the electric field $\vec{F}$. The polarization at a given time depends on the electric field at that time and the electric field at previous times. The history dependence leads to the well-known phenomenon of hysteresis, which is utilized in nonvolatile memory technology.

21.2 Syntax and implementation

DESSIS implements a model for ferroelectrics that features minor loop nesting and memory wipeout. Figure 15.70 demonstrates these properties and Section 21.3 on page 15.350 discusses them further. To activate the model, specify the keyword `Polarization` in the `Physics` section of the DESSIS command file. Use the optional parameter `Memory` to prescribe the maximum allowed nesting depth of minor loops. The smallest allowed value for `Memory` is 2; the default value is 10. If minor loop nesting becomes too deep, the nesting property of the minor loops can be lost. However, the polarization curve remains continuous.

For example:

```
Physics (region = "Region.17") {
   Polarization (Memory=20)
}
```

switches on the ferroelectric model in region `Region.17` and sets the size of the memory to 20 turning points for each element and each mesh axis.

To obtain a plot of the polarization field, specify `Polarization/Vector` in the `Plot` section of the DESSIS command file.

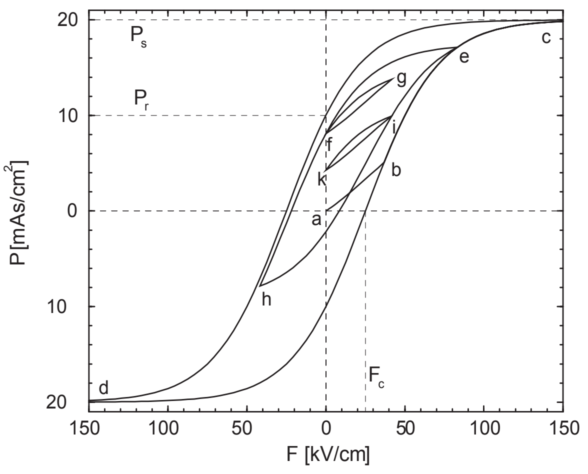


Figure 15.70 Example polarization curve

DESSIS characterizes the static properties of a ferroelectric material by three parameters: the remanent polarization P_r, the saturation polarization P_s, and the coercive field F_c. The hysteresis curve in Figure 15.70 on page 15.349 illustrates these quantities. Furthermore, DESSIS parameterizes the transient response of the ferroelectric material by the relaxation times τ_E and τ_P, and by a nonlinear coupling constant k_n (see Section 21.3).

Specify the values for these parameters in the `Polarization` section of the DESSIS parameter file, for example:

```
Polarization
{ * Remanent polarization P_r, saturation polarization P_s,
  * and coercive field F_c for x,y,z direction (crystal axes)
       P_r = (1.0000e-05, 1.0000e-05, 1.0000e-05) #[C/cm^2]
       P_s = (2.0000e-05, 2.0000e-05, 2.0000e-05) #[C/cm^2]
       F_c = (2.5000e+04, 2.5000e+04, 2.5000e+04) #[V/cm]
  * Relaxation time for the auxiliary field tau_E, relaxation
  * time for the polarization tau_P, nonlinear coupling kn.
       tau_E = (0.0000e+00, 0.0000e+00, 0.0000e+00) #[s]
       tau_P = (0.0000e+00, 0.0000e+00, 0.0000e+00) #[s]
       kn    = (0.0000e+00, 0.0000e+00, 0.0000e+00) #[cm*s/V]
}
```

The parameters in this example are the defaults for the material `InsulatorX`. For all other materials, all default values are zero. Each of the three numbers given for any of the parameters corresponds to the value for the respective coordinate axis of the mesh. If a `P_s` component is zero, the ferroelectric model is disabled along the corresponding direction. If a `P_s` component is nonzero, the respective `P_r` and `F_c` components must also be nonzero. Furthermore, the `P_r` component must be smaller than the `P_s` component. By default, the relaxation times are zero, which means that polarization follows the applied electric field instantaneously.

In devices with ferroelectric and semiconductor regions, it is sometimes difficult to obtain an initial solution of the Poisson equation. In many cases, the `LineSearchDamping` option can solve these problems. To use this option, start the simulation like:

```
coupled (LineSearchDamping=0.01) { Poisson }
```

See Section 2.9.1 on page 15.55 for details about this parameter.

21.3 Model description

The vector quantity $\vec{P}$ is split into its components along the main axes of the mesh coordinate system. This results in one to three scalar problems. DESSIS handles each problem separately using the model from [121] with extensions for transient behavior [122].

First, DESSIS computes an auxiliary field F_{aux} from the electric field F:

$$\frac{d}{dt}F_{aux}(t) = \frac{F(t) - F_{aux}(t)}{\tau_E} \qquad (15.445)$$

Here, τ_E is a material-specific time constant. For $\tau_E = 0$ or for quasistationary simulations, $F_{aux} = F$.

From the auxiliary field, DESSIS computes the auxiliary polarization P_{aux}. The auxiliary polarization P_{aux} is an algebraic function of the auxiliary field F_{aux}:

$$P_{aux} = c \cdot P_s \cdot \tanh(w \cdot (F_{aux} \pm F_c)) + P_{off} \qquad (15.446)$$

where P_s is the saturation polarization, F_c is the coercive field, and:

$$w = \frac{1}{2F_c}\log\frac{P_s + P_r}{P_s - P_r} \tag{15.447}$$

where P_r is the remanent polarization. In (Eq. 15.446), the plus sign applies to the decreasing auxiliary field and the minus sign applies to the increasing auxiliary field. The different signs reflect the hysteretic behavior of the material. P_{off} and c in (Eq. 15.446) result from the polarization history of the material, see below.

Finally, from the auxiliary polarization and auxiliary field, DESSIS computes the actual polarization P:

$$\frac{d}{dt}P(t) = \frac{P_{\text{aux}}[F_{\text{aux}}(t)] - P(t)}{\tau_P}\left(1 + k_n\left|\frac{d}{dt}F_{\text{aux}}(t)\right|\right) \tag{15.448}$$

Here, τ_P and k_n are material-specific constants. For $\tau_P = 0$ or for quasistationary simulations, $P = P_{\text{aux}}$.

Upper and lower turning points are points in the P_{aux}-F_{aux} diagram where the sweep direction of the auxiliary field F_{aux} changes from increasing to decreasing, and from decreasing to increasing, respectively. At each bias point, the most recent upper and lower turning points, (F_u,P_u) and (F_l,P_l), must both be on each of the two curves defined by (Eq. 15.446); this requirement determines P_{off} and c.

DESSIS ‘memorizes’ turning points as they are encountered during a simulation. The memory always contains (∞,P_s) as the oldest and $(-\infty,-P_s)$ as the second oldest turning point. By using (Eq. 15.446), these two points define a pair of curves with $c = 1$ and $P_{\text{off}} = 0$; together, the two curves form the saturation loop. All other pairs of turning points result in $c < 1$ and define a pair of curves forming minor loops.

When the auxiliary field leaves the interval defined by F_l and F_u of the two newest turning points, these two turning points are removed from the memory; this reflects the memory wipeout observed in experiments. The pair of turning points that are newest in the memory, after this removal, determines the further $P_{\text{aux}}(F_{\text{aux}})$ relationship.

As the older of the dropped turning points was originally reached by walking on the curve defined by the turning points that now (after dropping) again determine $P_{\text{aux}}(F_{\text{aux}})$, the polarization curve remains continuous. For example, see points e, f, and i in Figure 15.70 on page 15.349. Furthermore, the minor loop defined by the two dropped turning points is nested inside the minor loop defined by the present turning points. The nesting of minor loops is also a feature known from experiments on ferroelectrics. Figure 15.70 illustrates this by the loops f-g and i-k, both of which are nested in loop e-h, which in turn is nested in loop c-d.

In small-signal (AC) analysis (see Section 3.8.3 on page 15.117), a very small periodic signal is added to the DC bias. As a result, the (auxiliary) polarization at each point of the ferroelectric material changes along a very small minor loop nested in the main loop that stems from the DC variation of the bias voltage. The average slope of this minor loop is always smaller than the slope of the main loop at the point where the loops touch. Consequently, even at very low frequencies, the AC response of the system is different from what would be obtained by taking the derivative of the DC curves.

As an example of how the turning point memory works, see Figure 15.70. The points on the polarization curve are reached in the sequence a, b, c, d, e, f, g, f, h, i, k, i, e, c. For simplicity, a quasistationary process is assumed and, therefore, $F_{\text{aux}} = F$ and $P_{\text{aux}} = P$. Starting the simulation at point a, the newest point in memory is b (DESSIS initializes it in this way). The second newest point is a negative saturation point (l), and the oldest point is a positive saturation point (u). This memory state is denoted by [blu]. Therefore, the curve segment (that is, the coefficients c and P_{off}) from a to b is determined by the points l and b.

When crossing b and proceeding to c, b is dropped from the memory. As l is a saturation point, it is retained. Therefore, the memory becomes [lu]. These two points determine the curve from b to c. Turning at c, c is added to the memory, giving [clu]. From c to d, use c and l; at d, the memory becomes [dclu]; at point e, [edclu]; at f, [fedclu]; at g, [gfedclu]. Passing through f, the two newest points, f and g, are dropped and the memory is [edclu]; at h, [hedclu]; at i, [ihedclu]; at k, [kihedclu]. At i again, i and k are dropped, giving [hedclu]; at e, e and h are dropped, giving [dclu].

At the beginning of a simulation, the memory contains one turning point chosen such that the point $E_{\text{aux}} = 0$, $P_{\text{aux}} = 0$ is on the minor loop so defined (for example, point b in Figure 15.70 on page 15.349). The nature of the model is such that it is not possible to have a state of the system that is completely symmetric. In particular, even for a symmetric device and at the very beginning of the simulation, $P(E) \neq -P(-E)$. This asymmetry is most prominent for the virginal curves (for example, a-b in Figure 15.70) of the ferroelectric, which are different for different signs of voltage ramping.

CHAPTER 22 Mechanical stress effect modeling

22.1 Overview

Mechanical distortion of semiconductor microstructures results in a change in the band structure and carrier mobility. This effect is well known and appropriate computations of the change in the strain-induced band structure are based on the deformation potential theory [140]. The implementation of the deformation potential model in DESSIS is based on data and approaches presented in the literature [140]–[143]. Other approaches [106][182] implemented in DESSIS focus more on the description of piezoresistive effects.

Generally, the stress tensor $\bar{\sigma}$ is symmetric 3×3 matrices. Therefore, it only has six independent components. With the index transformation rule, it can be written in a six-component vector notation $\sigma_{11} \rightarrow \sigma_1, \sigma_{22} \rightarrow \sigma_2, \sigma_{33} \rightarrow \sigma_3, \sigma_{23} \rightarrow (\sigma_4)/2, \sigma_{13} \rightarrow (\sigma_5)/2, \sigma_{12} \rightarrow (\sigma_6)/2$ that simplifies tensor expressions. For example, to compute the strain tensor $\bar{\varepsilon}$ (which is needed for the deformation potential model), the generalized Hook's law for anisotropic materials is applied:

$$\varepsilon_i = \sum_{j=1}^{6} S_{ij}\sigma_j \tag{15.449}$$

where S_{ij} is the elasticity modulus (see Section 22.3 on page 15.354). In crystals with cubic symmetry such as silicon, the number of independent coefficients of the elasticity tensor (as other material property tensors) reduces to three by rotating the coordinate system parallel to the high-symmetric axes of the crystal [108]. This gives the following elasticity tensor $\bar{S}$:

$$\bar{S} = \begin{bmatrix} S_{11} & S_{12} & S_{12} & 0 & 0 & 0 \\ S_{12} & S_{11} & S_{12} & 0 & 0 & 0 \\ S_{12} & S_{12} & S_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & S_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & S_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 & S_{44} \end{bmatrix} \tag{15.450}$$

where the coefficients S_{11}, S_{12}, and S_{44} correspond to parallel, perpendicular, and shear components, respectively.

In DESSIS, the stress tensor can be defined in the stress coordinate system $(\vec{e}_1, \vec{e}_2, \vec{e}_3)$. To transfer this tensor to another coordinate system (for example, the crystal system $(\vec{e}'_1, \vec{e}'_2, \vec{e}'_3)$, which is a common operation), the following transformation rule between two coordinate systems is applied:

$$\sigma'_{ij} = a_{ik}a_{jl}\sigma_{kl} \tag{15.451}$$

where $\bar{a}$ is the rotation matrix:

$$a_{ik} = \left(\vec{e}'_i, \vec{e}_k\right) / \left(\left\|\vec{e}'_i\right\| \left\|\vec{e}_k\right\|\right) \tag{15.452}$$

22.2 Syntax and implementation

To activate any of the stress-dependent models described in this section, the keyword `Piezo` must be specified in the `Physics` section of the input command file. Optionally, it can contain stress component specifications (if stress is constant over the device or region), and the components $\vec{e}_1$ `OriKddX` and $\vec{e}_2$ `OriKddY` of the coordinate system where the stress is defined (see Table 15.134):

```
Physics {Piezo(
   Stress = (XX, YY, ZZ, YZ, XZ, XY)
   OriKddX = (1,0,0)
   OriKddY = (0,1,0)
   )
}
```

NOTE The stress system is always defined relative to the DESSIS simulation coordinate system (in the `Piezo` section of DESSIS input file). The simulation coordinate system is defined relative to the crystal orientation system by default but, in the DESSIS parameter file (see Section 22.3.1 on page 15.356), it is possible to define the crystal system relative to the simulation system. The default orientation of all coordinate systems is <100>.

Table 15.134 General keywords for Piezo

Parameter	Description
`Stress=(XX, YY, ZZ, YZ, XZ, XY)`	Specifies uniform stress [Pa] if the `Piezo` file is not given in the `File` section.
`OriKddX = (1,0,0)`	Defines Miller indices of the stress system relative to the simulation system.
`OriKddY = (0,1,0)`	Defines Miller indices of the stress system relative to the simulation system.
`Model(<options>)`	Selects stress-dependent models in `<options>` (see Section 22.3, Section 22.4 on page 15.357, and Section 22.5 on page 15.359).

There are two ways to define position-dependent stress values. In one option, a field of stress values [Pa] (as obtained by mechanical structure analysis) is read by specifying the `Piezo` entry in the `File` section:

```
File { ...
   Piezo = <piezofile>
}
```

Otherwise, a physical model interface can be used for stress specification. Optional parameters, which depend on the selected stress model, are described in the following sections.

NOTE Stress values in all these stress specifications should be in Pa (1 Pa = 10 dyn/cm^2) and tensile stress should be positive according to convention.

22.3 Deformation of band structure

In the deformation potential theory, the strains are considered to be relatively small. The change in energy of each carrier subvalley, caused by the deformation of the lattice, is a linear function of the strain. By default (for silicon), DESSIS considers three subvalleys for electrons (which are applied to three two-fold subvalleys in the conduction band) and two subvalleys for holes (which are applied to heavy-hole and light-hole subvalleys in the valence band). The number of carrier subvalleys can be changed in the DESSIS parameter file (see Section 22.3.1 on page 15.356). The following equation has been proposed [141] for the calculation of the change in the energy of carrier subvalleys:

$$\Delta E_{B,i} = \xi_{i1}^{B}(\varepsilon_{11}' + \varepsilon_{22}' + \varepsilon_{33}') + \xi_{i2}^{B}(\varepsilon_{11}' - \varepsilon_{33}') + \xi_{i3}^{B}(\varepsilon_{22}' - \varepsilon_{33}') + \xi_{i4}^{B}\varepsilon_{23}' + \xi_{i5}^{B}\varepsilon_{13}' + \xi_{i6}^{B}\varepsilon'_{12} \tag{15.453}$$

where `B` can be `C` or `V` (conduction or valence subvalleys), '`i`' corresponds to the carrier subvalley number, ξ_{ij}^{B} are the deformation potential constants, and ε_{ij}' are the components of the strain tensor in the crystal coordinate system (see Section 22.1 on page 15.353 for a description of tensor transformations).

Bir and Pikus [183] proposed another model for the strain-induced change in the energy of carrier subvalleys in silicon where they ignore the shear strain for electrons and suggest nonlinear dependence for holes:

$$\begin{aligned}
\Delta E_{C,i} &= \Xi_d(\varepsilon_{11}' + \varepsilon_{22}' + \varepsilon_{33}') + \Xi_u\varepsilon_{ii}' \\
\Delta E_{V,i} &= -a(\varepsilon_{11}' + \varepsilon_{22}' + \varepsilon_{33}') \pm \delta E \\
\delta E &= \sqrt{\frac{b}{2}((\varepsilon_{11}' - \varepsilon_{22}')^2 + (\varepsilon_{22}' - \varepsilon_{33}')^2 + (\varepsilon_{11}' - \varepsilon_{33}')^2) + d^2(\varepsilon_{11}'^2 + \varepsilon_{22}'^2 + \varepsilon_{33}'^2)}
\end{aligned} \tag{15.454}$$

where Ξ_d, Ξ_u, a, b, d are other deformation potentials that correspond to the Bir and Pikus model. The sign $\pm$ separates heavy-hole and light-hole subvalleys of silicon in (Eq. 15.454). Both (Eq. 15.453) and (Eq. 15.454) have the common part $(\varepsilon_{11}' + \varepsilon_{22}' + \varepsilon_{33}')$ and, therefore, these expressions were combined in one general expression that gives a flexibility of its definition in the DESSIS parameter file (see Section 22.3.1 on page 15.356):

$$\begin{aligned}
\Delta E_{B,i} &= \delta E_1 + \delta E_2 + \delta E_3 \\
\delta E_1 &= \xi_{i1}^{B}(\varepsilon_{11}' + \varepsilon_{22}' + \varepsilon_{33}') + \xi_{i2}^{B}(\varepsilon_{11}' - \varepsilon_{33}') + \xi_{i3}^{B}(\varepsilon_{22}' - \varepsilon_{33}') + \xi_{i4}^{B}\varepsilon_{23}' + \xi_{i5}^{B}\varepsilon_{13}' + \xi_{i6}^{B}\varepsilon'_{12} \\
\delta E_2 &= \xi_{i1}^{B2}\varepsilon_{11}' + \xi_{i2}^{B2}\varepsilon_{22}' + \xi_{i3}^{B2}\varepsilon_{33}' \\
\delta E_3 &= \xi_{i4}^{B2}\sqrt{\frac{(\xi_{i5}^{B2})^2}{2}((\varepsilon_{11}' - \varepsilon_{22}')^2 + (\varepsilon_{22}' - \varepsilon_{33}')^2 + (\varepsilon_{11}' - \varepsilon_{33}')^2) + (\xi_{i6}^{B2})^2(\varepsilon_{11}'^2 + \varepsilon_{22}'^2 + \varepsilon_{33}'^2)}
\end{aligned} \tag{15.455}$$

where ξ_{ij}^{B2} are deformation potentials that correspond to the Bir and Pikus model and ξ_{i4}^{B2} is a unitless constant that defines mainly a sign.

Using the stress tensor $\bar{\sigma}$ from the input file, DESSIS recomputes it from the stress coordinate system to the tensor $\bar{\sigma}'$ in the crystal system by (Eq. 15.451). The strain tensor $\bar{\varepsilon}'$ is a result of applying Hook's law (Eq. 15.449) to the stress $\bar{\sigma}'$. Using (Eq. 15.453) or (Eq. 15.454), the energy band change can be computed for each conduction and valence carrier subvalleys. DESSIS does not modify the effective masses, but uses the averaged values of conduction and valence bands, ΔE_C and ΔE_V:

$$\frac{\Delta E_C}{kT_{300}} = -\log\left[\frac{1}{n_C}\sum_{i=1}^{n_C}\exp\left(\frac{-\Delta E_{C,i}}{kT_{300}}\right)\right] \tag{15.456}$$

$$\frac{\Delta E_V}{kT_{300}} = \log\left[\frac{1}{n_V}\sum_{i=1}^{n_V}\exp\left(\frac{\Delta E_{V,i}}{kT_{300}}\right)\right] \tag{15.457}$$

where n_C and n_V are the number of subvalleys considered in the conduction and valence bands, respectively.

The band gap and affinity can be modified:

$$E_g = E_{g0} + \Delta E_C - \Delta E_V \tag{15.458}$$

$$\chi = \chi_0 - \Delta E_C \tag{15.459}$$

where the index ‘0’ corresponds to the affinity and band-gap values before stress deformation.

22.3.1 Syntax and implementation

To activate the deformation potential model, the name of the model must be specified in the `Piezo` section of the input file, for example:

```
Physics (Region = "StrainedSilicon") {
   Piezo(
      Model(DeformationPotential)
   )
}
```

All parameters of the model can be modified in the `LatticeParameters` section of the parameter file. The DESSIS simulation coordinate system relative to the crystal orientation system can be defined by the `X` and `Y` vectors in this section. The default is `X=[100], Y=[010]`. In addition, there is an option to represent the crystal system relative to the DESSIS simulation system. In this case, the keyword `CrystalAxis` must be in the `LatticeParameters` section, and the `X` and `Y` vectors will represent the <100> and <010> axes of the crystal system in the DESSIS simulation system.

The number of conduction and valence band subvalleys n_C and n_V are defined by `NC` and `NV` in the parameter file; by default, n_C=3, n_V=2. Appropriate deformation potential constants ξ_{ij}^C, ξ_{ij}^V, ξ_{ij}^{C2}, and ξ_{ij}^{V2} [eV] are defined in the fields `DC[i]`, `DV[i]`, `DC2[i]`, and `DV2[i]`of the parameter file, respectively.

Elasticity modulus S_{ij} [10^{-12} cm^2/dyn] can be specified in the field `S[i][j]` in the parameter file. If the cubic crystal system is selected, it is sufficient to specify S_{11}, S_{12}, and S_{44}. For a hexagonal crystal system, S_{33} and S_{13} must also be specified. In an arbitrary case, all elements of the upper triangle matrix must be specified. Otherwise, they are set to 0.

As an example, the following section represents silicon `LatticeParameters` defined for the Bir and Pikus model in the parameter file:

```
LatticeParameters{
* Crystal system, elasticity, and deformation potential are defined.
 X = (1, 0, 0)                #[1]
 Y = (0, 1, 0)                #[1]
 S[1][1] = 0.77               # [1e-12 cm^2/dyn]
 S[1][2] = -0.21              # [1e-12 cm^2/dyn]
 S[4][4] = 1.25               # [1e-12 cm^2/dyn]
 CrystalSystem = 0            # [1]
 NC = 3                       # [1]
 NV = 2                       # [1]
 DC(1) = -8.6,0,0,0,0,0 DC2(1) = 9.5,0,0,0,0,0
 DC(2) = -8.6,0,0,0,0,0 DC2(2) = 0,9.5,0,0,0,0
 DC(3) = -8.6,0,0,0,0,0 DC2(3) = 0,0,9.5,0,0,0
 DV(1) = -2.1,0,0,0,0,0 DV2(1) = 0,0,0,-1,0.5,4
 DV(2) = -2.1,0,0,0,0,0 DV2(2) = 0,0,0,1,0.5,4
}
```

22.4 Tensor-mesh piezoresistive option

Due to an applied mechanical stress, the mobility can become a tensor. The numeric approximation of the transport equations with tensor mobility is complicated (see Chapter 20 on page 15.339). However, if the mesh is a tensor one, the approximation is simpler. For this option, off-diagonal mobility elements are not used and, therefore, there are no mixed derivatives in the approximation. Such an approximation gives an M-matrix property for the Jacobian and it permits stable stress simulations. Very often, critical regions are simple and the mesh constructed in such regions can be close to a tensor one.

NOTE The off-diagonal elements of the mobility tensor appear only if there is a shear stress or the DESSIS simulation coordinate system is different from the crystal system.

This approach [106][107] focuses on the modeling of the piezoresistive effect. The basic part of the model is a linear extension of the constitutive current relations for electrons and holes in single-crystalline silicon [107] for small stresses:

$$\begin{aligned} \overline{\mu_\alpha} &= \mu_\alpha^0(\bar{1} - \overline{\Pi} \cdot \bar{\sigma}), \qquad \alpha = n, p \\ \vec{J}_\alpha &= \overline{\mu_\alpha} \cdot \left(\frac{\vec{J}_\alpha^0}{\mu_\alpha^0}\right) \end{aligned} \tag{15.460}$$

where $\overline{\mu_\alpha}$ is the second rank mobility tensor, μ_α^0 denotes the isotropic mobility without stress, $\bar{1}$ is the identity tensor, $\bar{\sigma}$ is the stress tensor, $\Pi\alpha$ is the tensor of piezoresistive coefficients that depends on the doping concentration and temperature distribution, and $\vec{J}_\alpha^0$ is the vector of the carrier current without the stress.

The stress tensor $\bar{\sigma}$ and mobility matrix $\overline{\mu_\alpha}$ are symmetric 3×3 matrices. Therefore, they only have six independent components.

With the index transformation rule (see Section 22.1 on page 15.353), they can be written in a six-component vector notation. In crystals with cubic symmetry such as silicon, the number of independent coefficients of the piezoresistance tensor reduces to three by rotating the coordinate system parallel to the high-symmetric axes of the crystal [108]:

$$\overline{\Pi} = \begin{bmatrix} \Pi_{11} & \Pi_{12} & \Pi_{12} & 0 & 0 & 0 \\ \Pi_{12} & \Pi_{11} & \Pi_{12} & 0 & 0 & 0 \\ \Pi_{12} & \Pi_{12} & \Pi_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & \Pi_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & \Pi_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 & \Pi_{44} \end{bmatrix} \tag{15.461}$$

Since the coordinate system of the simulation is not necessarily parallel to the high-symmetric axis of the crystal, the orientations of the x-axis and y-axis of the crystal system can be specified in the DESSIS command file (see Section 22.3.1 on page 15.356).

External strain leads to a change in the effective masses and anisotropic scattering. The first effect is described by an independent constant term $\Pi_{ij,\,kon}^{\alpha}$, but the second effect, the scattering, can be calculated [109] at room temperature for low-doping concentrations ($\Pi^{\alpha}_{ij,\,var}$) and multiplied by a doping-dependent and temperature-dependent factor $P_{\alpha}(N, T)$.

Both effects are considered in the piezoresistive coefficients by [109]:

$$\Pi^{\alpha}_{ij} = \Pi^{\alpha}_{ij,\,var}P_{\alpha}(N, T) + \Pi^{\alpha}_{ij,\,kon} \tag{15.462}$$

In the case of electrons, the scalar mobility used in the drift-diffusion and hydrodynamic equations is a mean value averaged over the different conduction band minima. If the symmetry of the crystal is destroyed, for example by external strain, the conduction band valleys shift and, therefore, yield electron transfers between the valleys. This redistribution of electrons in the conduction band leads to anisotropic scattering. In the case of holes, the mobility is an averaged quantity including heavy and light holes. External strain leads to a lift of the degeneracy at the valence band maximum.

According to the literature [109], the doping-dependent and temperature-dependent factor $P_{\alpha}(N, T)$ can be given by:

$$P_{\alpha}(N, T) = \frac{300}{T}\,\frac{F_{1/2}\left(\frac{E_{F,\,\alpha}}{kT}\right)}{F_{1/2}\left(\frac{E_{F,\,\alpha}}{kT}\right)} \tag{15.463}$$

where $F_{1/2}(x)$ and $F_{1/2}(x)$ are the Fermi integrals of the order `1/2` and its first derivative. The Fermi energy E_F is equal to $F_n - E_C$ for electrons and $E_V - F_p$ for holes. They are calculated by using appropriate analytic approximations [110] where the charge neutrality is assumed between carrier and doping (`N`), and it gives the doping dependence of the model. The numeric evaluation of $P_{\alpha}(N, T)$ is based on an analytic fit of the Fermi integrals [111].

The default values of the piezoresistive coefficients for low-doped silicon at 300 K are listed in Table 15.135 and Table 15.136 on page 15.359. They can be changed in the DESSIS command file as described in Section 22.4.1.

22.4.1 Syntax and implementation

The piezoresistive model is applied in a simulation by including the name of the model in the subsection `Model` of the `Piezo` section of the input command file. With the specification of the piezoresistive coefficients, this section appears as:

```
Physics {
   Piezo( Model(Mobility(Tensor))
   PiezoNkon = ($\Pi^n_{11,kon}$, $\Pi^n_{12,kon}$, $\Pi^n_{44,kon}$)
   PiezoNvar = ($\Pi^n_{11,var}$, $\Pi^n_{12,var}$, $\Pi^n_{44,var}$)
   PiezoPkon = ($\Pi^p_{11,kon}$, $\Pi^p_{12,kon}$, $\Pi^p_{44,kon}$)
   PiezoPvar = ($\Pi^p_{11,var}$, $\Pi^p_{12,var}$, $\Pi^p_{44,var}$)
   )
}
```

Table 15.135 Piezoresistive parameters for holes

$\Pi^p_{11,\,kon}$	$\Pi^p_{12,\,kon}$	$\Pi^p_{44,\,kon}$	$\Pi^p_{11,\,var}$	$\Pi^p_{12,\,var}$	$\Pi^p_{44,\,var}$	**Unit**
5.1	–2.6	28	1.5	1.5	110	$1\times10^{-12}\text{cm}^2\text{dyn}^{-1}$

Table 15.136 Piezoresistive parameters for electrons

$\Pi^n_{11,\,kon}$	$\Pi^n_{12,\,kon}$	$\Pi^n_{44,\,kon}$	$\Pi^n_{11,\,var}$	$\Pi^n_{12,\,var}$	$\Pi^n_{44,\,var}$	**Unit**
0	0	0	–102.6	53.4	–13.6	$1\times10^{-12}\text{cm}^2\text{dyn}^{-1}$

The `Mobility` keyword can have different options: `Tensor`, `eTensor`, and `hTensor`, where ‘`e`’ or ‘`h`’ switches on the stress-dependent mobility model only for electrons or holes, respectively. These options use only the constant piezoresistivity coefficients $\Pi^\alpha_{ij} = \Pi^\alpha_{ij,\,var} + \Pi^\alpha_{ij,\,kon}$ where the doping-dependent and temperature-dependent factor (Eq. 15.463) is equal to 1. To activate the doping and temperature dependence (Eq. 15.462), add the keyword `Kanda`, for example, `Tensor(Kanda)`.

NOTE This model can be used only with the keyword `TensorGridAniso` in the `Math` section.

To visualize the mobility multiplication tensor in (Eq. 15.460), the tensor $-\overline{\Pi} \cdot \bar{\sigma}$ can be plotted on the mesh nodes. This tensor is a symmetric 3×3 matrix and, therefore, has six independent values. To plot these values on the mesh nodes for electron and hole mobilities, there are the keywords `eMobilityStressFactorXX`, `eMobilityStressFactorYY`, `eMobilityStressFactorZZ`, `eMobilityStressFactorYZ`, `eMobilityStressFactorXZ`, `eMobilityStressFactorXY`, `hMobilityStressFactorXX`, `hMobilityStressFactorYY`, `hMobilityStressFactorZZ`, `hMobilityStressFactorYZ`, `hMobilityStressFactorXZ`, `hMobilityStressFactorXY`.

22.5 Strain-induced mobility model

Another approach [182] implemented in DESSIS focuses on the modeling of the mobility changes due to the carrier redistribution between subvalleys in silicon. As a known example, the electron mobility is enhanced in a strained-Si layer grown on top of a thick, relaxed SiGe. Due to the lattice mismatch (which can be controlled by the Ge mole fraction), the thin silicon layer appears to be ‘stretched’ (under biaxial tension).

The origin of the electron mobility enhancement can be explained [182] by considering the six-fold degeneracy in the conduction band. The biaxial tensile strain lowers two perpendicular valleys (Δ_2) with respect to the four-fold in-plane valleys (Δ_4). Therefore, electrons are redistributed between valleys and Δ_2 is occupied more heavily. It is known that the perpendicular effective mass is much lower than the longitudinal one. Therefore, this carrier redistribution and reduced intervalley scattering enhance the electron mobility. The hole depends on the strain mainly due to redistribution of holes between light and heavy valleys, and changes the effective masses in these valleys.

The model consistently accounts for a change of subvalley energy as described in Section 22.3 on page 15.354, that is, a modification of the deformation potentials in (Eq. 15.455) will affect the strain-Si mobility model. In the crystal coordinate system, the model gives only the diagonal elements of the electron mobility matrix but, for holes, the mobility is still isotropic.

The following expressions have been suggested [182] for the electron and hole mobilities:

$$
\begin{aligned}
\mu_{ii}^{n} &= \mu_{n}^{0}\left[1+\frac{1-m_{nl}/m_{nt}}{1+2(m_{nl}/m_{nt})}\left(\frac{F_{1/2}\left(\frac{F_n-E_C-\Delta E_{C,i}}{kT}\right)}{F_{1/2}\left(\frac{F_n-E_C-\Delta E_C}{kT}\right)}-1\right)\right]\\
\mu^{p} &= \mu_{p}^{0}\left[1+\left(\frac{\mu_{pl}^{0}}{\mu_{p}^{0}}-1\right)\frac{(m_{pl}/m_{ph})^{1.5}}{1+(m_{pl}/m_{ph})^{1.5}}\left(\frac{F_{1/2}\left(\frac{E_V+\Delta E_{V,l}-F_p}{kT}\right)}{F_{1/2}\left(\frac{E_V+\Delta E_{V,h}-F_p}{kT}\right)}-1\right)\right]
\end{aligned}
\qquad (15.464)
$$

where:

μ_n^0 and μ_p^0 are electron and hole mobility models without the strain.

m_{nl} and m_{nt} are the electron longitudinal and transfer masses in the subvalley.

$\Delta E_{C,i}$ and ΔE_C are computed by (Eq. 15.455) and (Eq. 15.456), respectively.

The index '`i`' corresponds to a direction (for example, μ_{11}^n is the electron mobility in the direction of `x`-axis of the crystal system and, therefore, $\Delta E_{C,1}$ should correspond to the two-fold subvalley along the `x`-axis).

μ_{pl}^0 is the mobility of light holes without the strain.

m_{pl} and m_{ph} are the hole light and heavy masses.

$\Delta E_{V,l}$ and $\Delta E_{V,h}$ are computed also by (Eq. 15.455) with the specification of light-hole and heavy-hole subvalleys numbers in the parameter file (see Section 22.5.1 on page 15.361).

F_n and F_p are quasi-Fermi levels of electrons and holes.

NOTE The carrier quasi-Fermi levels, as for (Eq. 15.463), are computed assuming the charge neutrality between carrier and doping, and it gives the doping dependence of the model.

In the case of Boltzmann statistics, (Eq. 15.464) is simplified and the dependence on the carrier (doping) concentration disappears:

$$
\begin{aligned}
\mu_{ii}^{n} &= \mu_{n}^{0}\left[1+\frac{1-m_{nl}/m_{nt}}{1+2(m_{nl}/m_{nt})}\left(\exp\left(\frac{\Delta E_C-\Delta E_{C,i}}{kT}\right)-1\right)\right]\\
\mu^{p} &= \mu_{p}^{0}\left[1+\left(\frac{\mu_{pl}^{0}}{\mu_{p}^{0}}-1\right)\frac{(m_{pl}/m_{ph})^{1.5}}{1+(m_{pl}/m_{ph})^{1.5}}\left(\exp\left(\frac{\Delta E_{V,l}-\Delta E_{V,h}}{kT}\right)-1\right)\right]
\end{aligned}
\qquad (15.465)
$$

22.5.1 Syntax and implementation

The strain-induced mobility model can be activated regionwise or materialwise with the following keyword in the `Mobility` statement of `Piezo` model:

```
Physics {
   Piezo( Model(Mobility(Subband)) )
}
```

The keyword `Subband` assumes Boltzmann statistics and, in this case, (Eq. 15.465) is used. To activate the doping dependence and (Eq. 15.464), the following keywords should be used: `Subband(Egley)`. The prefixes '`e`' and '`h`' can be added if it is necessary to activate the model separately for electrons and holes. For example, it can be `eSubband(Egley)`. The model parameters (see (Eq. 15.464)) can be specified in the DESSIS parameter file in the section `StressMobility` as follows:

```
StressMobility{
 me_lt = 4.81            # [1]
 elec_100 = 1            # [1]
 elec_010 = 2            # [1]
 elec_001 = 3            # [1]
 mh_lh = 0.32653         # [1]
 mobh_l = 2.79           # [1]
 hole_light = 1          # [1]
 hole_heavy = 2          # [1]
}
```

where `me_lt` is the ratio m_{nl}/m_{nt}, `mh_lh` is the ratio m_{pl}/m_{ph}, and `mobh_l` is the ratio μ^0_{pl}/μ^0_p. Numbers `elec_100`, `elec_010`, and `elec_001` correspond to indexes of deformation potentials in the `LatticeParameters` section (see Section 22.3.1 on page 15.356) for two-fold electron subvalleys in the direction `[100]`, `[010]`, and `[001]`, respectively. The numbers `hole_light` and `hole_heavy` correspond to indexes of light-hole and heavy-hole subvalleys in the same `LatticeParameters` section.

NOTE The model ignores the off-diagonal elements of the mobility matrix that can appear with a transformation of this matrix due to a rotation of the coordinate system from the crystal to the simulation one. This model can be used only with the keyword `TensorGridAniso` in the `Math` section.

To visualize the mobility multiplication tensor in (Eq. 15.464), the tensor with the subtracted unit tensor can be plotted on the mesh nodes. This tensor is a symmetric 3×3 matrix and, therefore, has six independent values. To plot these values on the mesh nodes for electron and hole mobilities, there are the keywords `eMobilityStressFactorXX`, `eMobilityStressFactorYY`, `eMobilityStressFactorZZ`, `eMobilityStressFactorYZ`, `eMobilityStressFactorXZ`, `eMobilityStressFactorXY`, `hMobilityStressFactorXX`, `hMobilityStressFactorYY`, `hMobilityStressFactorZZ`, `hMobilityStressFactorYZ`, `hMobilityStressFactorXZ`, `hMobilityStressFactorXY`.

CHAPTER 23 Galvanic transport model

23.1 Syntax and implementation

In the `Physics` section of the DESSIS command file, specify the magnetic field vector using the keyword `MagneticField = (<x>,<y>,<z>)`. In the following example, a field of 0.1 Tesla is applied parallel to the z-axis:

```
Physics { ...
   MagneticField = (0.0, 0.0, 0.1)
}
```

In addition, the `EdgeMagneticDiscretization` option must be specified in the `Math` section and `Newdiscretization` must not be used:

```
Math { ...
   EdgeMagneticDiscretization
   -Newdiscretization
}
```

NOTE The galvanic transport model cannot be combined with the hydrodynamic model or impact ionization model.

23.2 Model description

For analysis of magnetic field effects in semiconductor devices, the transport equations governing the flow of electrons and holes in the interior of the device must be set up and solved.

To this end, the commonly used drift-diffusion-based model of the carrier current densities $\vec{J}_n$ and $\vec{J}_p$ must be augmented by magnetic field–dependent terms that account for the action of the transport equations governing the flow of electrons and holes in the interior of the device *Lorentz force* on the motion of the carriers [103]–[105]:

$$\vec{J}_\alpha = -\underline{\sigma}_\alpha \nabla\phi_\alpha - \underline{\sigma}_\alpha \frac{1}{1+(\mu^*{}_\alpha B)^2}[\mu^*{}_\alpha \vec{B}\times\nabla\phi_\alpha + \mu^*{}_\alpha \vec{B}\times(\mu^*{}_\alpha \vec{B}\times\nabla\phi_\alpha)] \quad \text{with} \quad \alpha = n, p \qquad (15.466)$$

Here, $\underline{\sigma}_\alpha$ denotes the electric conductivity of carrier type α, $\mu^*{}_\alpha$ is the Hall mobility, ϕ_α is the quasi-Fermi potential, and $\vec{B}$ is the magnetic induction. With a nonzero magnetic field, the electric conductivities are tensors of rank 2 with two principal components parallel and perpendicular to the vector $\vec{B}$. In terms of the concentrations of electrons and holes (n and p) and the respective drift mobilities (μ_n and μ_p), the electric conductivities are given by the relations $\underline{\sigma}_n = qn\mu_n$ and $\underline{\sigma}_p = qp\mu_p$. Here, q is the elementary charge. The perpendicular (transverse) components of Hall and drift mobility are related by $\mu^*{}_n = r_n\mu_n$ and $\mu^*{}_p = r_p\mu_p$, where r_n and r_p denote the Hall scattering factors. In the case of bulk silicon, typical values are $r_n = 1.1$ and $r_p = 0.7$.

CHAPTER 24 Thermal properties

24.1 Heat capacity

Table 15.137 lists the values of the heat capacity used in the simulator.

Table 15.137 Values of heat capacity c for various materials

Material	c [J/K cm^3]	Reference
Silicon	1.63	[92]
Ceramic	2.78	[42]
SiO_2	1.67	[42]
Poly Si	1.63	≈ Si

If the user wants a constant lattice heat capacity without touching the parameter file, specify `HeatCapacity(Constant)` in the `Physics` section of the input file.

24.2 Temperature-dependent lattice heat capacity

The temperature dependence of the lattice heat capacity is modeled by the empirical function:

$$c = cv+cv\_b*T+cv\_c*T^2+cv\_d*T^3$$

The equation coefficients can be specified in the parameter file by using the syntax:

```
LatticeHeatCapacity{
    cv = 1.63           # [J/(K cm^3)]
    cv_b = 0.0000e+00   # [J/(K^2 cm^3)]
    cv_c = 0.0000e+00   # [J/(K^3 cm^3)]
    cv_d = 0.0000e+00   # [J/(K^3 cm^3)]
}
```

By default, DESSIS uses these values from the parameter file. To switch to this default model, specify `HeatCapacity(TempDep)`.

NOTE All coefficients of the model in the parameter file can be mole dependent for mole-dependent materials.

24.3 Thermal conductivity

DESSIS uses the following temperature-dependent thermal conductivity κ in silicon [93]:

$$\kappa(T) = \frac{1}{a + bT + cT^2} \qquad (15.467)$$

where a = 0.03 cm K/W, b = 1.56×10^{-3} cm/W, and c = 1.65×10^{-6} cm/WK. The range of validity is 200 K $\leq$ T to well above 600 K. A graphical representation of κ for silicon is given in Figure 15.71. Values of the thermal conductivity for further materials are given in Table 15.138.

Table 15.138 Values of thermal conductivity κ of silicon versus temperature

Material	**κ [W/(cm K)]**	**Reference**
Silicon	(Eq. 15.467)	[93]
Ceramic	0.167	[94]
SiO_2	0.014	[92]
Poly Si	1.5	≈ Si

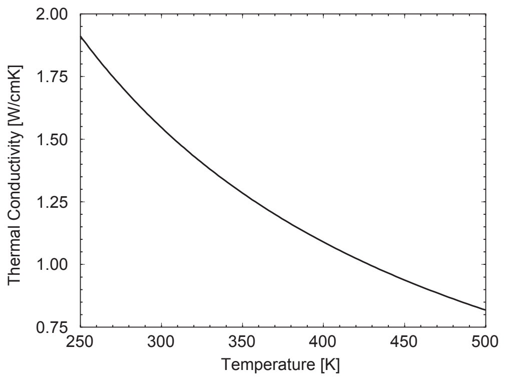


Figure 15.71 Thermal conductivity κ of silicon versus temperature

24.4 Temperature-dependent thermal conductivity

As additional options to the standard specification of the thermal conductivity model, there are two different expressions to define either thermal resistivity $\chi = 1/k$ or thermal conductivity for any material. This is performed by using `Formula` in the parameter file or special keywords in the input file.

For `Formula=0` (thermal resistivity specification), DESSIS uses:

χ = `1/kappa+1/kappa_b*T+1/kappa_c*T`^{2}

For `Formula=1` (thermal conductivity specification), it is:

k = `kappa+kappa_b*T+kappa_c*T`^{2}

Using the following syntax in the parameter file, the expressions can be switched coefficients specified:

```
Kappa{
   Formula = 0
      1/kappa = 0.03          # [K cm/W]
      1/kappa_b = 1.5600e-03  # [cm/W]
      1/kappa_c = 1.6500e-06  # [cm/(W K)]
      kappa = 1.5             # [W/(K cm)]
      kappa_b = 0.0000e+00    # [W/(K^2 cm)]
      kappa_c = 0.0000e+00    # [W/(K^3 cm))]
}
```

The `Physics` section of the input file provides more flexibility to switch these expressions by using the keywords:

```
ThermalConductivity{
   TempDep Conductivity (Formula = 1)
   Constant Conductivity (Formula = 1 without temperature dependence)
   TempDep Resistivity (Formula = 0)
   Constant Resistivity (Formula = 0 without temperature dependence)
}
```

By default, DESSIS uses `Formula` specified in the parameter file.

NOTE All these coefficients in the parameter file can be mole dependent for mole-dependent materials.

24.5 Thermoelectric power (TEP)

Theoretically, the electron and hole absolute thermoelectric powers P_n and P_p for nondegenerate semiconductors can be written as [95][96]:

$$P_n = -\kappa_m \frac{k_B}{q}\left[\left(\frac{5}{2} - s_n\right) + \ln\left(\frac{N_C}{n}\right)\right] \tag{15.468}$$

$$P_p = \kappa_p \frac{k_B}{q}\left[\left(\frac{5}{2} - s_p\right) + \ln\left(\frac{N_V}{p}\right)\right] \tag{15.469}$$

To use these expressions, specify `AnalyticTEP` in the `Physics` section of the DESSIS command file. The coefficients in these equations are available in the `TEPower` parameter set in the DESSIS parameter file (see Table 15.139).

Table 15.139 Coefficients for thermoelectric power

Symbol	Parameter name	Default
κ_n	scale_n	1
κ_p	scale_p	1
s_n	s_n	1
s_p	s_p	1

Without the keyword `AnalyticTEP`, DESSIS uses a table of experimental values of the TEPs for silicon published by Geballe and Hull [97] as a function of temperature and carrier concentration. DESSIS extrapolates P_nT and P_pT linearly for temperatures between 360 K and 500 K, thus preserving the 1/T dependence of data presented at higher temperatures by Fulkerson et al. [98], which holds up to near the intrinsic temperature.

P_n and P_p are shown in Figure 15.72 as a function of temperature and carrier concentration as used in DESSIS.

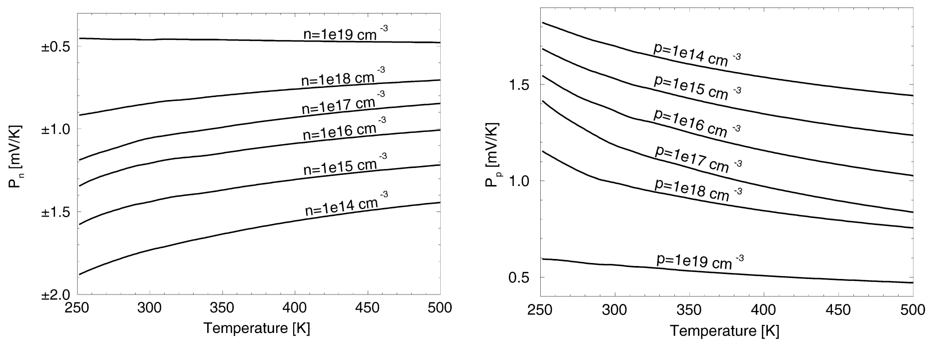


Figure 15.72 TEPs Pn (*left*) and Pp (*right*) as a function of temperature and carrier concentration

Part III Physics of Lasers and Light-Emitting Diodes

This part of the DESSIS manual contains the following chapters:

CHAPTER 25 Introduction to lasers and LEDs

DESSIS includes optional models for the comprehensive simulation of semiconductor lasers and light-emitting diodes (LEDs). Both edge-emitting lasers and vertical-cavity surface-emitting lasers (VCSELs) are supported. Drift-diffusion or hydrodynamic transport equations for the carriers, the Schrödinger equation for quantum well gain, modal optical rate equations, and the Helmholtz equation are solved self-consistently in the quasistationary and transient modes.

Spontaneous and stimulated optical recombinations are calculated in the active and bulk regions according to Fermi's golden rule. They are added as carrier recombination mechanisms in the continuity equations and as modal gain and spontaneous emission in the photon rate equations. Different line-width broadening models are available for gain broadening.

Both bulk and quantum well (QW) lasers can be simulated. In the case of QW lasers, the optical polarization dependence of the optical matrix element is automatically taken into account. The QW subbands are calculated as the solution of the single-band Schrödinger equation in the effective mass approximation, assuming a box-shaped potential or by a multiple-bands k.p method. Both zinc-blende and wurtzite crystal structures can be treated. Strain effects can also be taken into account. The distribution of carriers in the well is determined according to the quantum mechanical wavefunctions and QW density of states.

In the unquantized direction, drift-diffusion transport applies to the carriers. In the quantized direction, two transport models are available. The simple transport model assumes thermionic emission at the heterointerfaces, which form the quantum well. The advanced transport model separates the QW carrier distributions into a bound and continuum distribution. The fraction of bound and continuum carriers is then self-consistently computed by additional scattering equations, mainly contributed by carrier-carrier and carrier-optical phonon scatterings.

The next section is a short introduction on how to set up a laser simulation with single and dual grids, and which type of output can be extracted from the simulation. The theoretical foundations of the laser simulator with detailed equations are discussed, followed by a description of other useful features of the laser and LED simulator. Finally, there is a discussion on how to simulate different types of lasers and LED.

25.1 Overview

A laser simulation is different from a silicon device simulation in three aspects:

- The Helmholtz equation must be solved to determine the optics of the device.
- A photon rate equation must be included to couple the electronics to the optics.
- The carrier-scattering processes at the quantum well must be treated differently because it is no longer in the drift-diffusion regime. In this case, a new syntax must be introduced to activate the laser simulation within the DESSIS framework. These new syntaxes are highlighted as the anatomy of the command file used for laser or LED simulations is examined in the next sections.

The procedure for performing a laser simulation is similar to that for a silicon device simulation. Figure 15.73 illustrates the procedure.

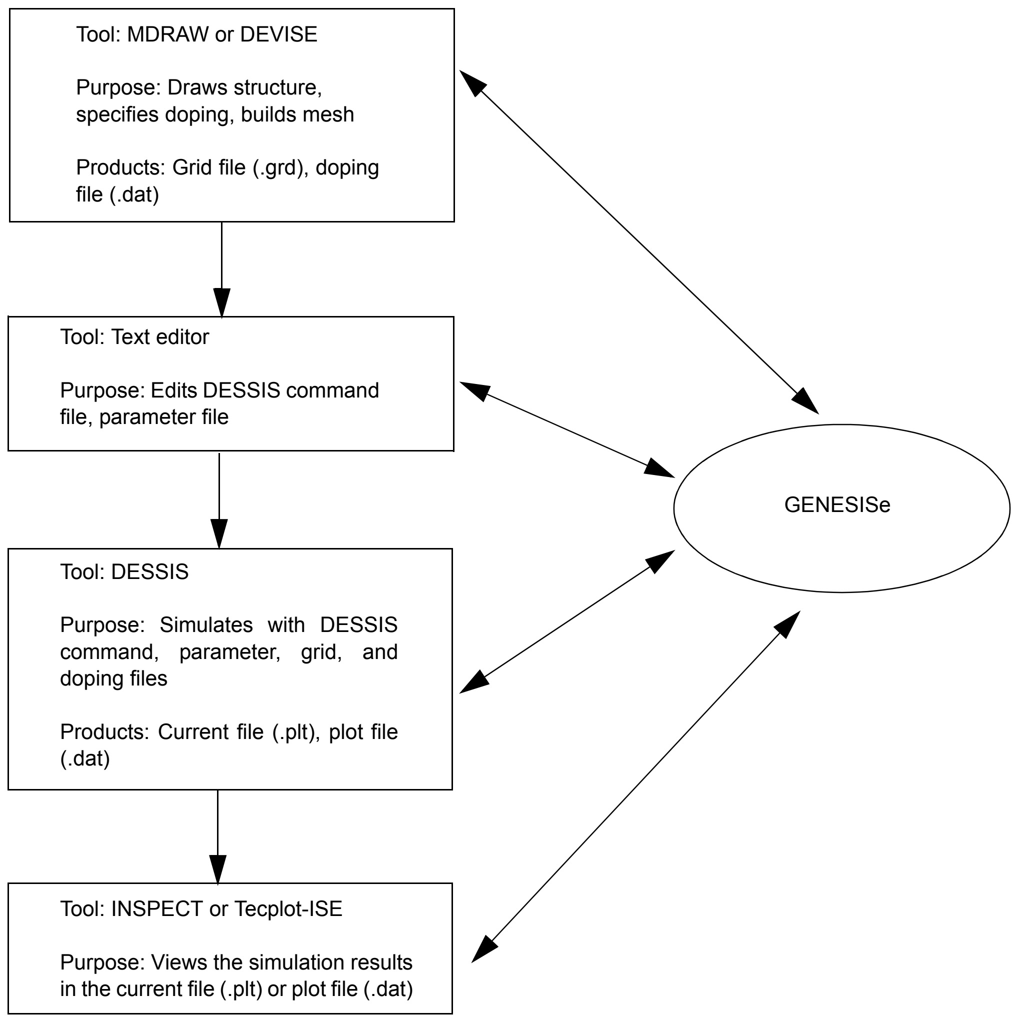

Tool: MDRAW or DEVISE
Purpose: Draws structure, specifies doping, builds mesh
Products: Grid file (.grd), doping file (.dat)
Tool: Text editor
Purpose: Edits DESSIS command file, parameter file
Tool: DESSIS
Purpose: Simulates with DESSIS command, parameter, grid, and doping files
Products: Current file (.plt), plot file (.dat)
Tool: INSPECT or Tecplot-ISE
Purpose: Views the simulation results in the current file (.plt) or plot file (.dat)
GENESISe

Figure 15.73 Flow of tasks in a single laser simulation run

First, the structure and doping profile are drawn in the graphics editor, MDRAW or DEVISE (refer to the MDRAW and DEVISE manuals). Second, MDRAW or DEVISE generates the mesh and the doping information on the mesh, and saves them in the grid file (with extension `.grd`) and doping file (with extension `.dat`), respectively. Instead of drawing the structure laboriously, the user has the option to write a script to construct the structure, and build the mesh and doping automatically. In this way, it is possible to parameterize any device dimension, material composition, and doping profile easily. This feature is used by GENESISe to control batch jobs and automatically optimize user-specified performance benchmarks through parameter variation.

Next, the user can use any editor to edit the DESSIS command file and parameter file. The parameter file contains a complete list of default material parameter models such as band gap, thermal conductivities, recombination parameters, traps parameters, effective masses, and mobilities. The user can edit any of these material parameters. The DESSIS command file contains instructions to run the simulation. If the user runs a 1D, 2D, or 3D simulation, only the grid and doping files will change. The DESSIS command file and parameter file are unchanged.

After the simulation is completed, DESSIS saves the final results in the current file (with extension `.plt`) and the plot file (with extension `.dat`). These results can be viewed and plotted in INSPECT and Tecplot-ISE, respectively. If users specify some other feature such as far-field computation, additional files are produced that can be viewed in INSPECT or Tecplot-ISE, depending on the file type.

All the abovementioned steps can be controlled from GENESISe, which can also schedule jobs over the network and extract specified results of the simulation. In this manner, automation can be achieved for device optimization. Refer to the GENESISe manual for more information.

25.2 Command file syntax

The DESSIS command file permits full control of how a simulation will run. This includes choosing the type of numeric solver, the physics to be included, and the equations to couple. This fundamental framework provides a platform with unlimited expansion possibilities for new features, new models, and new devices. The fastest way to understand the command file syntax is to look at a generic example that users can copy and modify for their specific needs. Two generic examples are presented for typical laser and LED simulations.

NOTE The character '#' in the DESSIS command file means that any text proceeding from '#' is treated as a remark or as a preprocessor directive.

25.2.1 Single-grid edge-emitting laser simulation

This generic example uses the same grid for both the optical and electrical problems. The optical problem is defined by the optics solver and the electrical problem is defined by the semiconductor transport equations.

```
# ----- Dessis command file for laser simulation -----

# ----- Specify initial bias conditions of the contacts -----
Electrode {
   { Name="p_Contact" voltage=0.8 AreaFactor = 200}
   { Name="n_Contact" voltage=0.0 AreaFactor = 200}
}

# ----- Tell Dessis where to read/save the parameters/results -----
File {
   # ----- Specify grid and doping files -----
   Grid = "mesh_mdr.grd"
   Doping = "mesh_mdr.dat"

   # ----- Specify the material parameter file -----
   Parameters = "des_las.par"

   # ----- Specify where to output various results. The presence
   # of these keywords also activates the particular save option -----
   Current = "current"
   Output = "log"
   Plot = "multiqw_plot"

   # ----- Option to save gain curves -----
# ModeGain = "gain"

   # ----- Option to save optics -----
# SaveOptField = "laserfield"
# VCSELNearField = "vcselfield"              # for VCSELs
  OptFarField = "farfield"                   # Activate farfield computation

   # ----- Option to load in externally generated optical field -----
# OptField0 = "external_mode_0"
# OptField1 = "external_mode_1"
}

# ------- Can extract virtually any variable of the simulation ------
Plot {
   # ----- Normal dessis variables -----
   Doping
   xMoleFraction yMoleFraction
   DonorConcentration AcceptorConcentration
   ElectronAffinity
   BandGap
   ConductionBandEnergy ValenceBandEnergy
   EffectiveIntrinsicDensity
   eDensity hDensity
   eMobility hMobility
   eQuasiFermi hQuasiFermi
   eGradQuasiFermi hGradQuasiFermi
   eCurrent hCurrent TotalCurrent
   eCurrent/vector hCurrent/vector TotalCurrent/vector
   ElectricField
   Potential SpaceCharge
   SRH Auger
   RefractiveIndex
   Dielectric/Element
   # ----- Extra variables for laser simulation -----
   QWeDensity QWhDensity
   QWeQuasiFermi QWhQuasiFermi
   MatGain
   LaserIntensity
   OpticalIntensityMode0
   OpticalIntensityMode1
   OpticalPolarizationAngleMode0
   OpticalPolarizationAngleMode1
}

Physics {
   AreaFactor = 2               # for symmetric simulation
   # ------ Keyword Laser activates the laser simulation ------
   Laser(
          # ----- Specify the optical solver to use -----
          Optics(
                 # ----- Finite element vectorial solver -----
                 # The scalar solver is activated using FEScalar
                 FEVectorial(EquationType = Waveguide        # or Cavity
                            Symmetry = Symmetric              # or NonSymmetric or Periodic
                            LasingWavelength = 800            # [nm]
                            TargetEffectiveIndex = (3.4 3.34) # initial guess
                            #Polarization = (TE TM)           # for scalar solver only
                            #AzimuthalExpansion = (0 1)       # for VCSELs
                            Boundary = ("Type2" "Type1")      # choose boundary conditions
                            ModeNumber = 2                    # up to 10 modes
                 )
          )

#        VCSEL(NearField(10.0 0 50)) # for VCSELs
         TransverseModes               # for edge emitter simulation
         CavityLength = 900            # [micron]
         lFacetReflectivity = 0.9      # Left facet power reflectivity
         rFacetReflectivity = 0.4       # Right facet power reflectivity
         OpticalLoss = 10.0             # [1/cm]
         WaveguideLoss                  # activate feedback of loss from Optics

         # ------- Choice of Gain broadening functions, choose only one -------
#        Broadening (Type=Landsberg Gamma=0.010)         # Gamma in [eV]
#        Broadening (Type=CosHyper Gamma=0.010)
         Broadening (Type=Lorentzian Gamma=0.010)

         # ----- Nonlinear Gain saturation model -----
#        Broadening(Type = LorentzianSat
#                    Gamma = 4.39e-4                    # Broadening [eV]
#                    eIntrabandRelTime = 1e-13          # electron relaxation time
#                    hIntrabandRelTime = 1e-13          # hole relaxation time
#                    PolarizationRelTime = 3e-12        # polarization relaxation time
#        )

         # ----- Specify Quantum Well physics -----
         qwTransport
         qwExtension = AutoDetect       # auto read of QW widths
         qwScatmodel
         QWeScatTime = 8e-13            # [s]
         QWhScatTime = 4e-13            # [s]
         eQWMobility = 9200             # [cm^2/Vs]
         hQWMobility = 400              # [cm^2/Vs]

         # ----- Include QW Strain effects -----
         Strain

         # ----- Can scale stim and spon gain independently -----
         StimScaling = 1.0
         SponScaling = 1.0

         # ----- Specify dependency of refractive index ----
         RefractiveIndex(TemperatureDep CarrierDep)
   )

   # ----- User specified physics of transport -----
   Thermionic                    # thermionic emission over interfaces
   HeteroInterfaces              # allow discontinuous bandgap & quasi-Fermi level
   Mobility ( DopingDep )
   Recombination ( SRH Auger )
   EffectiveIntrinsicDensity ( NoBandGapNarrowing )
   Fermi

   # ------ Option to turn on temperature simulation ------
# Thermodynamic
# Hydrodynamic
# RecGenHeat
}

# ----- Can specify the physics of each region separately -----
Physics (region="pbulk") { MoleFraction(xfraction=0.28) }
Physics (region="nbulk") { MoleFraction(xfraction=0.28) }
Physics (region="psch") { MoleFraction(xfraction=0.09) }
Physics (region="nsch") { MoleFraction(xfraction=0.09) }
Physics (region="barr") { MoleFraction(xfraction=0.09) }

# ----- Quantum Wells -----

Physics (region="qw1") {
   MoleFraction( xfraction = 0.8 Grading=0.00 )
   Active       # keyword to indicate active region
```

```
}

Physics (region="qw2") {
   MoleFraction( xfraction = 0.8 Grading=0.00 )
   Active
}

# ----- Choice and control of numerical methods -----
Math {
   Digits = 5
   Extrapolate
   Method = pardiso     # pardiso is a parallel sparse solver
   ErReff(electron) = 1.e3
   ErReff(hole) = 1.e3
   Iterations = 30
   Notdamped = 50
   RelErrControl
   ElementEdgeCurrent
# Cylindrical          # in case of cylindrical symmetry

}

# ----- Solver part, specify what to couple and solve -----
Solve {
   # ----- Get initial guesses, coupled means Newton’s iteration -----
   Poisson
   coupled {Hole Electron QWhScatter QWeScatter Poisson }
   coupled {Hole Electron QWhScatter QWeScatter Poisson PhotonRate}

   # ----- Ramping the voltage -----
   quasistationary (
        # ----- Specify ratio step size of voltage ramp -----
        InitialStep = 0.001
        MaxStep = 0.05
        Minstep = 1e-5

        # ----- Save plot variables at intervals of simulation -----
#       Plot { range=(0,1) intervals=1}

        # --- Plotting the gain vs photon energy ---
#       PlotGain { range=(0,1) intervals=5}
#       PlotGainPara{range=(1.22,1.32) intervals=150}

        # --- Plotting the VCSEL transverse nearfield ---
#       VCSELNearField { range=(0,1) intervals=1}

        # --- Plotting the far field -----
        PlotFarField{range=(0,1) intervals=1}
        PlotFarFieldPara{range=(40,60) intervals=40 Scalar1D Scalar2D Vector2D}

        # --- Save the optical field ---
#       SaveOptField { range=(0,1) intervals=1}

        # ----- Specify the final voltage ramp goal -----
        Goal {name="p_Contact" voltage=1.8})
        {
           # ----- Gummel iterations for self-consistency of Optics -----
           Plugin(BreakOnFailure){
                # --- Newton iterations for coupled equations -----
                 Coupled { Electron Hole Poisson QWeScatter QWhScatter
                        PhotonRate }
                 Optics
                 Wavelength
```

```
                  }
            }

        # ----- At 1.8V, perform transient simulation -----
        transient (
              InitialTime = 0.0
              FinalTime = 1.0e-6      # [s]
              InitialStep = 1e-5
              MaxStep = 1e-3
              MinStep = 1e-8 )
              {
                 Plugin(BreakOnFailure){
                      Coupled { Electron Hole Poisson QWeScatter QWhScatter
                                PhotonRate }
#                     Optics
#                     Wavelength
                 }
              }
}
```

To elaborate on some remarks in this example of code:

- The general skeleton for the command file contains the `Electrode`, `File`, `Plot`, `Math`, `Physics`, and `Solve` sections. Within each section, additional subsections can be defined and, therefore, provide an object-oriented approach for defining the various parameters of the device and simulation.
- In the `Electrode` section, `AreaFactor` gives the scaling factor to account for the device width in the longitudinal direction (1 micron by default).
- In the `File` section, the keyword of a specific option must be included to activate that option. For example, DESSIS will only save the optical field if `SaveOptField` is included.
- In the `Plot` section, a more complete list of the standard DESSIS variables can be found in Section 2.6 on page 15.52. The laser simulation–specific plot variables are listed in the following sections.
- In the `Physics-Laser` section, `Laser` and `LED` simulations share these common options. The main difference between the different types of laser simulation is the choice of optical solver. In this example, the `FEVectorial` (finite element vectorial) solver is used. Other optical solver choices include `FEScalar`, `RayTrace`, `TMM1D`, and `EffectiveIndex`. The different types of optical solver are described in Chapter 27 on page 15.399.
- In the `Physics-Laser` section, users can select different gain-broadening functions. The gain-broadening functions, nonlinear gain saturation model, and quantum well parameters are described in Chapter 28 on page 15.439.
- In the `Physics` section, there are options to switch on the thermodynamic or hydrodynamic systems. To solve for the temperatures, it is necessary to define the thermode and couple the relevant equation in the command file. (This is not shown in this example.) Details of how to do this are in Section 29.7 on page 15.481.
- In the `Math` section, there is a selection of numeric engines including `pardiso`, `ils`, `super`, `umf`, `slip`. Users are encouraged to use different engines to find out which is the fastest and most accurate for their applications (see Section 2.10 on page 15.73).
- In the `Solve-quasistationary` section, the step size is expressed as a number between 0 and 1, which is mapped to the actual voltage ramping goal. In this case, the device was initially at 0.8 V and the goal was set to 1.6 V. Therefore, the unitless step of [0,1] maps to [0.8,1.6] V in the voltage ramp.

- In the `Solve-quasistationary` section, the keyword `Coupled` ensures that the listed equations are solved using Newton iterations. To enforce self-consistency of other systems (in this case, the `Wavelength` and `Optics`) in a Gummel-type iteration scheme, the `Plugin` feature was used. If it is not expected that the wavelength or optics will change as a function of the bias, they can be commented out in a similar way to what is done in the `Transient` statement.

25.2.2 Dual-grid edge-emitting laser simulation

In some cases, users may require different mesh sizes for the optical and electrical problems. As a general rule, the discretization for the optical mesh should be at least 20 points per wavelength for an accurate solution, but such fine discretization may not be necessary for the electrical problem. In this case, the dual-grid capability is invoked. This capability is derived from the circuit mixed-mode feature of DESSIS where the coupling of a few devices can be simulated in a self-consistent manner. The syntax of how this is performed is:

```
# ----- Dessis command file for dual grid laser simulation -----
File {
    Output = "dual_log"
}

# ----- Choice and control of the numerical method for the mixed-mode circuit -----
Math {
   # ----- Differences in a dual grid -----
   NoAutomaticCircuitContact
   DirectCurrentComputation
   Method = blocked
   Submethod = pardiso
   # ----- The rest are the same as the single grid -----
   Digits = 5
   Extrapolate
   ErReff(electron) = 1.e3
   ErReff(hole) = 1.e3
   Iterations = 30
   Notdamped = 50
   RelErrControl
   ElementEdgeCurrent
}

# ===== Define the Optical grid =====
# ----- Use keyword OpticalDevice -----
OpticalDevice optgrid {

   File {
      # ----- Read in the optical mesh -----
      Grid = "optmesh_mdr.grd"
      Doping = "optmesh_mdr.dat"
      Parameters = "des_las.par"
   }
   Plot {
      LaserIntensity
      OpticalIntensityMode0
   }
   # ----- Material region physics -----
   Physics (region="pbulk") { MoleFraction(xfraction=0.28) }
   Physics (region="nbulk") { MoleFraction(xfraction=0.28) }
   Physics (region="psch") { MoleFraction(xfraction=0.09) }
   Physics (region="nsch") { MoleFraction(xfraction=0.09) }
   Physics (region="barr") { MoleFraction(xfraction=0.09) }
   # ----- Quantum wells -----
   Physics (region="qw1") {
      MoleFraction( xfraction = 0.8 Grading=0.00 )
      Active
   }
   Physics (region="qw2") {
      MoleFraction( xfraction = 0.8 Grading=0.00 )
      Active
   }

}
# ===== End of Optical grid definition =====

# ===== Define the electrical grid and solver info =====
# ----- Use keyword Dessis -----
Dessis electricaldev {

   Electrode {
      {Name="p_Contact" voltage=0.8 AreaFactor = 200}
      {Name="n_Contact" voltage=0.0 AreaFactor = 200}
   }
   File {
      # ----- Read in electrical grid mesh -----
      Grid = "elecmesh_mdr.grd"
      Doping = "elecmesh_mdr.dat"
      Parameters = "des_las.par"

      Current = "elec_current"
      Plot =    "elec_plot"
      SaveOptField = "laserfield"
      ModeGain = "gain"
   }
   Plot {
      # ----- Similar to single grid, can include a long list -----
      LaserIntensity
      OpticalIntensityMode0
   }
   Physics {
       AreaFactor = 2        # takes device symmetry into account
       # ------ Laser definition, similar to single grid ------
       Laser(
          Optics(
                 FEVectorial(EquationType = Waveguide
                            Symmetry = Symmetric
                            LasingWavelength = 800
                            TargetEffectiveIndex = (3.4 3.32)
                            Boundary = ("Type2" "Type1")
                            ModeNumber = 2
               )
         )

         TransverseModes
         CavityLength = 900             # [micron]
         lFacetReflectivity = 0.9       # Left facet power reflectivity
         rFacetReflectivity = 0.4       # Right facet power reflectivity
         OpticalLoss = 10.0             # [1/cm]
         WaveguideLoss                  # activate feedback of loss from Optics
         # ----- Choose Gain broadening, similar to single grid -----
         Broadening (Type=Lorentzian Gamma=0.10)        # [eV]
           # ----- Specify QW parameters, similar to single grid -----
         qwTransport
         qwExtension = AutoDetect       # auto read QW widths
         qwScatmodel
```

```
            QWeScatTime = 8e-13            # [s]
            QWhScatTime = 4e-13            # [s]
            eQWMobility = 9200             # [cm^2/Vs]
            hQWMobility = 400              # [cm^2/Vs]
            # ----- QW Strain effects -----
            Strain
            # ----- Can scale stim and spon gain independently -----
            StimScaling = 1.0
            SponScaling = 1.0
            # ----- Specify dependency of refractive index ----
            RefractiveIndex(TemperatureDep CarrierDep)
         )
         # ----- Specify transport physics, similar to single grid -----
         Thermionic
         HeteroInterfaces
         Mobility ( DopingDep )
         Recombination ( SRH Auger )
         EffectiveIntrinsicDensity ( NoBandGapNarrowing )
         Fermi
      }
      # ----- Material region physics -----
      Physics (region="pbulk") { MoleFraction(xfraction=0.28) }
      Physics (region="nbulk") { MoleFraction(xfraction=0.28) }
      Physics (region="psch") { MoleFraction(xfraction=0.09) }
      Physics (region="nsch") { MoleFraction(xfraction=0.09) }
      Physics (region="barr") { MoleFraction(xfraction=0.09) }

      # ----- Quantum wells -----
      Physics (region="qw1") {
         MoleFraction( xfraction = 0.8 Grading=0.00 )
         Active
      }
      Physics (region="qw2") {
         MoleFraction( xfraction = 0.8 Grading=0.00 )
         Active
      }

   }
   # ===== End of electrical grid definition =====

   # ===== Define the circuit mixed-mode system =====
   System {
      # ----- Define opt1 of type optgrid -----
      optgrid opt1 ()
      # ----- Define d1 of type electricaldev, and coupled to opt1 -----
      electricaldev d1 (p_Contact=vdd n_Contact=gnd) {Physics{OptSolver="opt1"}}
      # ----- Set the initial bias voltage to circuit contacts -----
      Vsource_pset drive(vdd gnd){ dc=0.8 }
      Set ( gnd = 0.0 )
   }

   # ===== Solver part, slightly different from single grid =====
   Solve {
      Poisson
      # ----- Addition of Contact & Circuit, different from single grid -----
      coupled { Hole Electron QWhScatter QWeScatter Poisson Contact Circuit }
      coupled { Hole Electron QWhScatter QWeScatter Poisson Contact Circuit PhotonRate }

      quasistationary (
           # ----- Define ratio step size, similar to single grid -----
           InitialStep = 0.001
           MaxStep = 0.05
           Minstep = 1e-5
           # ----- Plot various quantities, similar to single grid -----
           Plot { Range=(0,1) Intervals=1 }
           PlotGain { range=(0,1) intervals=5 }
           PlotGainPara { range=(1.22,1.32) intervals=150 }
           SaveOptField { Range=(0,1) Intervals=1 }
           # ----- Specify final voltage, syntax different from single grid -----
           Goal { Parameter=drive.dc Value=1.6 })
           {
              # ----- Gummel iterations, similar to single grid -----
              Plugin (BreakOnFailure) {
                   Coupled { Electron Hole Poisson Contact Circuit
                              QWeScatter QWhScatter PhotonRate }
                   Optics
                   Wavelength
              }
           }
   }
```

This example uses the same device structure as in Section 25.2.1 on page 15.373, so that users can compare the single-grid and dual-grid simulations more clearly. Some important differences and similarities are:

- The optical and electrical grids may have different material-region physics, but the structural shape should be the same. This is to ensure that the interpolating of parameters and variables between the two grids is accurate. In this example, the electrical and optical devices have the same material regions, but the meshing resolutions are different, leading to two different grids.
- The control of the laser physics is encompassed in the definition of the electrical grid or device. The keyword to define the electrical grid or device is `Dessis`; `OpticalDevice` defines the optical grid.
- In the `System` section, `opt1` is defined as a device instance of type `optgrid` and `d1` is defined as a device instance of type `electricaldev`. The optical device instance `opt1` is then coupled as a circuit element to the electrical device instance `d1` by specifying `OptSolver="opt1"`. In addition, `p_Contact` and `n_Contact` of the electrical device are given the new names of `vdd` and `gnd`, respectively, in this coupled circuit (see Section 3.4 on page 15.106 for more information about the `System` command).
- In the `Solve` section, the keywords `Contact` and `Circuit` in the coupled statement ensure that self-consistency between the optical and electrical devices in the circuit is imposed.

25.2.3 Default output from laser or LED simulation

If the option `Current = "current"` is specified in the `File` section of the command file, DESSIS produces the current file at the end of the simulation. The current file (`current_des.plt`) contains laser-specific result variables as a function of bias. These result variables are:

- Time [s]
- `p_Contact` – `OuterVoltage` [V], `InnerVoltage` [V], `DisplacementCurrent`, `eCurrent`, `hCurrent`, `Charge`, and `TotalCurrent` [A]
- `n_Contact` – Same set of result variables as `p_Contact`
- `Mode0` – `TotalPower` [W], `PowerLeft`, `PowerRight`, `OpticalGain` [1/cm], `OpticalLoss` [1/cm], `SpontEmission` [1/cm], `Wavelength` [nm], `EffectiveIndex`, `OptConfinementFactor`
- `LEDWavelength` [nm] – Average wavelength of an LED emission

- `LEDEfficiency` – Extraction efficiency from an LED simulation
- `LEDOutput` [W] – Output power from an LED simulation

These result variables can be plotted using INSPECT. For example, by assigning `p_Contact->TotalCurrent` to the x-axis, `Mode0->TotalPower` to the left y-axis, and `p_Contact->OuterVoltage` to the right y-axis, the L–I–V curves for the laser diode are obtained as shown in Figure 15.74.

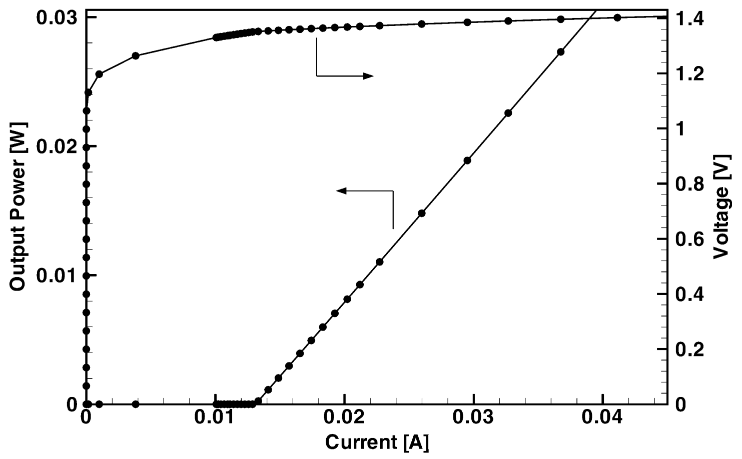


Figure 15.74 INSPECT plot of optical output power versus drive current

Scripts to automatically extract the threshold current, the slope efficiency, and so on are available. Users can refer to the ISE TCAD library or contact ISE Technical Support to obtain these scripts.

25.2.4 Plot variables specific to laser or LED simulations

If the option `Plot = "plot"` is specified in the `File` section of the command file, DESSIS creates the plot file at the end of the simulation. The plot file (`plot_des.dat`) contains the plot variables that the user has specified in the `Plot` statement, and these plot variables are given in DF–ISE format on the vertices of the mesh. The plot variable names are usually meaningful to make it easier to identify what they represent. Apart from the list of standard plot variables from a DESSIS simulation (see Appendix E on page 15.619), the `Laser` option introduces an additional set of plot variables:

`LaserIntensity`	Combined laser intensity for multimode lasing
`Dielectric`	Dielectric profile
`RefractiveIndex`	Refractive index profile
`MatGain`	Local material gain
`QWeDensity`, `QWhDensity`	Quantum well electron and hole bound states densities
`QWeQuasiFermi`, `QWhQuasiFermi`	Quantum well electron and hole quasi-Fermi levels
`OpticalIntensityMode0...9`	Optical intensity of each mode from `mode0` to `mode9`
`OpticalPolarizationAngleMode0...9`	Polarization of the optical field vector from `mode0` to `mode9`

By inputting the grid file and plot file into Tecplot-ISE, users can visualize the results. For example, Figure 15.75 shows the `LaserIntensity` from an edge-emitting laser simulation.

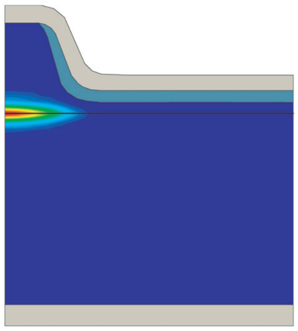


Figure 15.75 Two-dimensional contour plot of laser intensity

In addition, there is a list of options that users can switch on to obtain specific output files. These options are discussed in the next sections. Some options were included in the example in Section 25.2.1 on page 15.373, so that users understand how to activate these options. A summary of these options is:

- Gain curves – Gain versus energy at different biases
- Band structure plots – Results of k.p band structure and wavefunction calculations in quantum well regions
- Far-field plots – Far-field patterns at different biases
- Optical field vector and intensity – Optical intensities at different biases
- VCSEL near field – Transverse VCSEL near field at different biases
- LED radiation – Far zone radiation of an LED emission at different biases and different wavelengths

CHAPTER 26 Theoretical foundations of laser or LED simulation

26.1 Overview

Simulating a laser diode is one of the most complex problems in device simulation. The fundamental set of equations used in a laser simulation is:

- Poisson equation
- Carrier continuity equations
- Lattice temperature equation and hydrodynamic equations
- Quantum well scattering equations (for QW carrier capture)
- Quantum well gain calculations (Schrödinger equation)
- Photon rate equation
- Helmholtz equation

The first three equations are semiconductor transport equations for the drift-diffusion regime (see Section 4.2 on page 15.127). Other than the semiconductor transport equations, which solve the electrical drift-diffusion problem for holes and electrons, the carrier capture in the quantum well (QW) and the gain calculations are specific to the laser problem, and they link the electronics and optics. The QW carrier capture and gain calculations are discussed in Chapter 28 on page 15.439. The solution of the Helmholtz equation provides the optical modes and other optical quantities, and this is presented in Chapter 27 on page 15.399. Finally, the 'moderator' between the electronics and optics is the photon rate equation, which solves for the total number of photons. In the next section, the relationship between all these equations is shown and how to achieve self-consistency between all these equations is explained.

26.2 Coupling between optics and electronics

The coupling between the different equations in the laser simulation is illustrated in Figure 15.76.

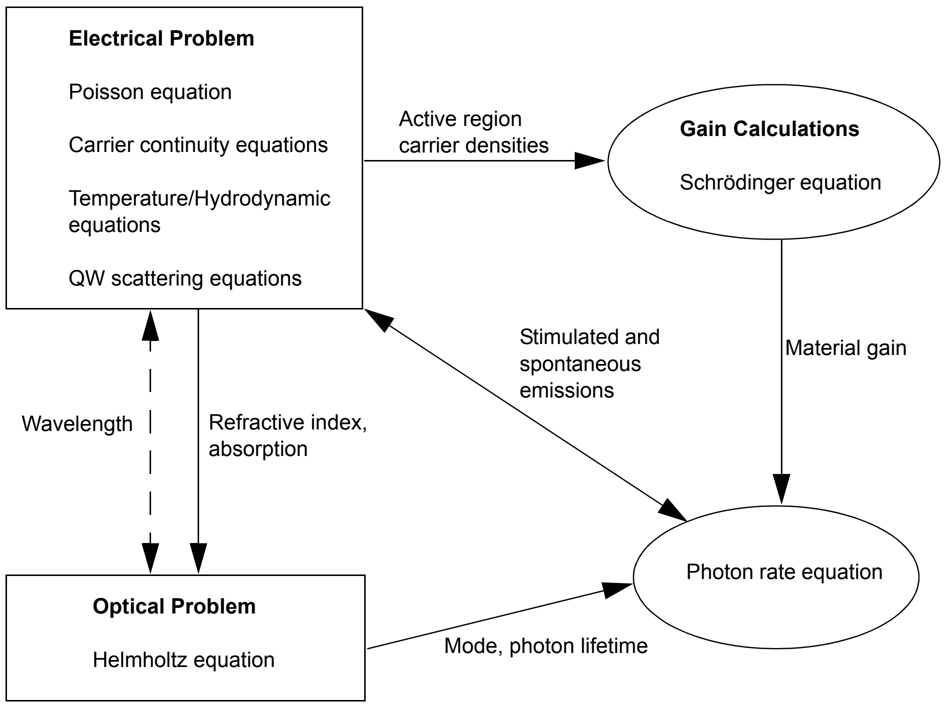


Figure 15.76 Coupling between the semiconductor transport and optics equations

The complexity of the laser problem is apparent from this relational chart. The key quantities exchanged between different equations are placed alongside the directional flows between the equation blocks. The optical problem must be separated from the electrical problem. The optical problem solves the Helmholtz equation and feeds the mode and photon lifetimes back to the set of active vertices. As a result, the coupling between the optics and electronics becomes nonlocal, and this leads to convergence problems if the Newton method is used to couple these two problems. Therefore, a Gummel iteration method (instead of a coupled Newton iteration) is required to couple the electrical and optical problems self-consistently.

The solution of the electrical problem provides the required refractive index changes and absorption to the optical solver, which solves for the modes. In the case of the Fabry–Perot edge-emitting laser, the wavelength is computed from the peak of the gain curve and fed into the optical problem. In VCSELs and distributed Bragg reflector (DBR) lasers, the wavelength is computed inside the optical resonance problem and is an input to the electrical problem instead.

The gain calculations involve the solution of the Schrödinger equation for the subband energy levels and wavefunctions. These quantities are used with the active-region carrier statistics to compute the optical matrix element in the material gain. The formulation of the material gain is based on Fermi's golden rule. For more details about the gain calculations, see Chapter 28 on page 15.439.

The photon rate equation takes the material gain and mode information to compute the modal gain and, subsequently, the stimulated and spontaneous recombination rates. These optical recombinations increase the photon population but reduce the carrier population. Therefore, these recombination rates must be added to the carrier continuity equations to ensure the conservation of particles (see Section 26.3 on page 15.388).

26.2.1 Algorithm for coupling electrical and optical problems

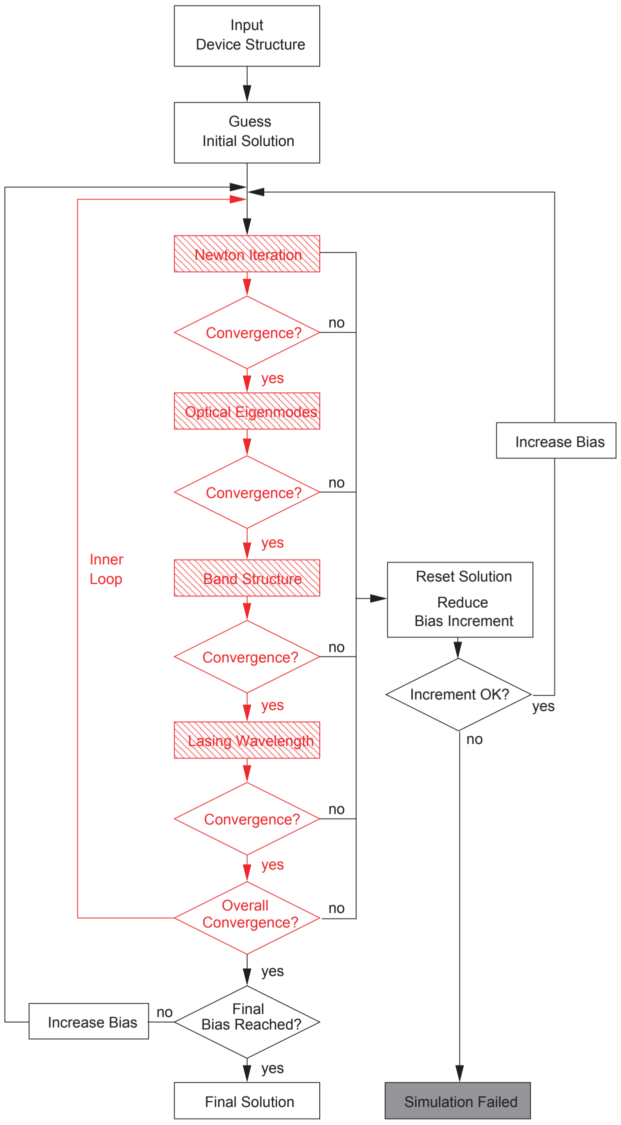


Figure 15.77 Algorithm flowchart for the self-consistent solution of laser equations

The Newton iteration is performed in DESSIS using the `Coupled` keyword, while the Gummel iteration is activated using the keyword `Plugin`. Figure 15.77 shows the algorithm flowchart for the laser simulation. In addition, refer to the `Solve` section in Section 25.2.1 on page 15.373 to see how the syntax is written.

The statement `Coupled {Electron Hole Poisson QWeScatter QWhScatter PhotonRate}` activates the Newton iterations for the electron and hole continuity equations, the Poisson equation, the QW carrier capture equations, and the photon rate equation. This is indicated by the Newton iteration block in Figure 15.77.

After the convergence of the Newton iterations, the updated refractive index profile (if it changes with temperature and carrier densities) is passed to the optical solver to solve for the eigen optical modes. At this instance, the band structures are also computed. In the case of VCSELs and DBR lasers, the resonant (or lasing) wavelength is also computed by the optical solver. The keyword `Plugin` ensures that the electrical and optical solutions are iterated self-consistently, that is, a Gummel-type iteration. Through this self-consistency, all the physics should be satisfied. DESSIS automatically handles the flow of the result variables between the electrical and optical solvers. There are also additional checkpoints to restrict the solution of each part from diverging during the Gummel iteration, as shown in Figure 15.77 on page 15.387.

When convergence is attained for the Gummel iteration (`Plugin`), the solution set for this bias is used as the initial guess for the next bias. In this way, the continuous wave operation of the laser diodes can be simulated.

26.3 Photon rate equation

The origin of the photon rate equation can be traced to the famous work of Henry [134] on the theory of spontaneous emission noise in open resonators. The cavity of an edge-emitting Fabry–Perot laser is considered an open electromagnetic resonator, and the modes are driven by the radiative recombination processes.

From the Maxwell equations, the wave equation for the electric field is derived as:

$$\nabla\times\nabla\times\mathcal{E} = -\mu_0\left(\sigma\frac{\partial\mathcal{E}}{\partial t} + \varepsilon_0\varepsilon_r\frac{d^2\mathcal{E}}{dt^2} + \frac{d^2P}{dt^2}\right) \tag{15.470}$$

where $\mathcal{E}$ is the electric field, σ is the conductivity, and ε_r is the relative permittivity. The source term P can be identified as the polarization due to the spontaneous recombination of electron–hole pairs. For the electric field, the following separation ansatz is made:

$$\mathcal{E}(x, y, z, t) = \Psi(x, y;t)\sqrt{S(t)}e^{i\varphi(t)}e^{i(\omega_0 t - \kappa_0 z)} + cc \tag{15.471}$$

where cc denotes the complex conjugate. The optical field, Ψ, is complex; while the photon density, $S(t)$, and phase factor, φ, are real quantities. Ψ is normalized according to:

$$\iint|\Psi|^2 dxdy = 1 \tag{15.472}$$

Inserting (Eq. 15.471) into (Eq. 15.470) results in the Helmholtz equation for $\Psi(x, y)$:

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\Psi + k_0^2(n^2(x, y) - \varepsilon_{eff})\Psi = 0 \tag{15.473}$$

which is discussed in Chapter 27 on page 15.399; a phase rate equation:

$$\frac{\partial\varphi}{\partial t} + \frac{\omega_0\mathcal{E}^{re}}{2\bar{\varepsilon}_r}\varphi = 0 \tag{15.474}$$

which is not included in DESSIS and, finally, the photon rate equation for $S(t)$:

$$\frac{\partial S}{\partial t} - (G(\omega) - L)S = \frac{c}{\varepsilon_r}\beta T^{sp}(\omega) \tag{15.475}$$

where β is the spontaneous emission factor that can be defined by the user (using keyword `SponEmiss`) and its default value is 1. The photon rate equation contains the modal gain, $G(\omega)$, the optical loss, L_{opt}, and the modal spontaneous emission, $T^{sp}(\omega)$.

These parameters are defined as:

$$G(\omega) = \iint r^{st}(x, y, E_\omega)|\Psi(x, y)|^2 dxdy \tag{15.476}$$

$$T^{sp}(\omega) = \iint r^{sp}(x, y, E_\omega)|\Psi(x, y)|^2 dxdy \tag{15.477}$$

$$L_{opt} = \iint \alpha(x, y)|\Psi(x, y)|^2 dxdy + Loss_{bg} + Loss_{cavity} + Loss_{waveguide} \tag{15.478}$$

The stimulated emission coefficient $r^{st}(x, y, E_\omega)$ and the spontaneous emission coefficient $r^{sp}(x, y, E_\omega)$ are computed locally at each active vertex from Fermi's golden rule, and their values are taken at the lasing energy E_ω. These radiative emission coefficients are discussed in Chapter 28 on page 15.439. The local optical intensity $|\Psi(x, y)|^2$ is solved from the Helmholtz equation. In the simulation, the spatial integrations are performed in the active regions of the laser only.

The total optical loss L_{opt} contains the loss due to local free carrier absorption $\alpha(x, y)$, the cavity loss $Loss_{cavity}$, the background optical loss $Loss_{bg}$, and the waveguide loss $Loss_{waveguide}$. The free carrier loss is described in detail in Section 29.1 on page 15.471. The user specifies the background loss. The waveguide loss (imaginary part of propagation constant) is fed back from the optical solver to the photon rate equation, and this includes the losses due to leakage waves from the waveguide.

The cavity loss for a Fabry–Perot cavity mainly contains the mirror loss and is given by:

$$Loss_{cavity} = \frac{1}{2L}\ln\left(\frac{1}{r_o r_1}\right) \tag{15.479}$$

where L is the cavity length and $r_o r_1$ are the facet power reflectivities. This formula is used mainly in the simulation of edge-emitting lasers. For VCSELs, the cavity loss is more complicated due to scattering and diffraction effects, and can only be accurately computed by a vectorial cavity solution. In this case, the net loss in the VCSEL photon rate equation, $(L_{opt} - G(\omega))$, must be solved from the optics equation when gain-guiding effects are included. This is elaborated in Section 26.5 on page 15.392.

Each optical mode corresponds to a distinct photon rate equation. For example, if a user specifies up to ten modes in a laser simulation, DESSIS automatically solves up to ten photon rate equations. The stimulated and spontaneous emissions from each photon rate equation are then summed to give a total radiative emission rate. This total rate is then added to the carrier continuity equation as a radiative recombination to ensure the conservation of electron–hole pair recombination and photon emission.

At lasing threshold, the modal gain of the laser saturates to the value of the optical losses, which results in a singularity in the steady-state photon rate:

$$S = \left(\frac{\bar{\varepsilon}_r}{c}\right) \cdot \frac{\beta T^{sp}(\omega)}{L_{opt} - G(\omega)} \tag{15.480}$$

A small fluctuation of the QW carrier densities and, therefore, gain during Newton iterations leads to large changes in the photon rate. In cases where the gain exceeds the loss, the photon rate suddenly becomes negative, which is nonphysical and pushes the entire Newton iteration towards divergence. To solve this problem, a slack variable, λ_{slack}, is introduced to constrain $(L_{opt} - G(\omega))$ to be always positive:

$$\lambda_{slack}^2 = L_{opt} - G(\omega) \tag{15.481}$$

so that the photon rate will always be positive. This important numeric technique to laser simulation was first proposed by Smith [184] and, in DESSIS, it is called the photon stabilization equation. When the photon rate equation is activated, DESSIS automatically activates the photon stabilization equation as well.

Table 15.140 Keywords for photon rate equation

Feature	Keyword	Example syntax

Photon rate equation	`PhotonRate`	`Solve {...`
		`   Coupled {PhotonRate ...}`
		`}`

Spontaneous emission factor	`SponEmiss=<float>`	`Physics {...`
		`   Laser (...`
		`      SponEmiss = 1.0`
		`   )`
		`}`

Cavity length [microns]	`CavityLength=<float>`	`Physics {...`
		`   Laser (...`
		`      Cavitylength = 500`
		`      lFacetReflectivity = 0.3`
		`      rFacetReflectivity = 0.99`
		`      OpticalLoss = 10.0`
		`      WaveguideLoss`
		`   )`
		`}`

Left-facet power reflectivity	`lFacetReflectivity=<float>`	
Right-facet power reflectivity	`rFacetReflectivity=<float>`	
Background optical loss [1/cm]	`OpticalLoss=<float>`	
Activates the feedback of waveguide loss from optical solver to the photon rate equation. Default is no feedback.	`WaveguideLoss`	

26.4 Waveguide optical modes and Fabry–Perot cavity

In an edge-emitting laser with a Fabry–Perot type cavity, the longitudinal direction is invariant. In this case, the optical propagation in the longitudinal z-direction can always be described by forward and backward traveling waves with characteristic $\exp(\pm j\beta z)$, where β is the longitudinal propagation constant.

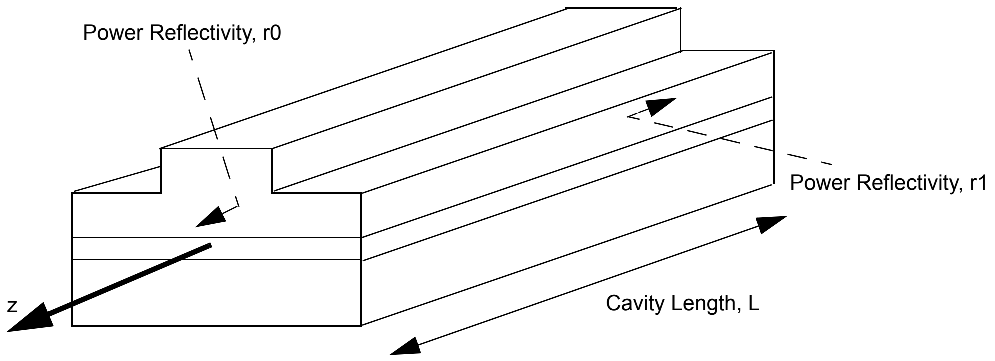


Figure 15.78 Waveguide problem in Fabry–Perot cavity assuming longitudinal invariance in z-direction

Therefore, the vector wave equation reduces to the Helmholtz equation:

$$(\nabla_t \times \nabla_t \times \quad - \nabla_t(\varepsilon^{-1}\nabla_t \cdot \varepsilon) + (\omega^2\mu_0\varepsilon(x, y) - \beta^2)) \cdot \boldsymbol{E}_t(x, y) = \boldsymbol{0} \qquad (15.482)$$

where $\boldsymbol{E}_t(x, y)$ is the transverse vectorial electric field, ω is the angular frequency, ε is the permittivity, β is the complex propagation constant, and ∇_t is the transverse gradient operator. The optical eigenmodes are the waveguide modes, which are characterized by the mode profile and propagation constant, β.

In a weakly guiding waveguide where the refractive index inhomogeneity, $\nabla_t(\varepsilon^{-1}\nabla_t \cdot \varepsilon)\boldsymbol{E}_t(x, y)$, is small, the vector Helmholtz equation reduces to its scalar form:

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\Psi + k_0^2(n^2(x, y) - \varepsilon_{eff})\Psi = 0 \tag{15.483}$$

where the scalar wavefunction $\Psi(x, y)$ describes either the TE-polarized or TM-polarized optical field component, and ε_{eff} is the effective dielectric constant. DESSIS solves the vectorial and scalar Helmholtz equations by the finite element method [135].

In the waveguide problem, the input parameters are:

- Wavelength (encompassed in $\lambda = (2\pi c)/\omega$)
- Refractive index profile
- Boundary conditions

The output is:

- Optical field profile
- Effective index (real part of propagation constant, $Re(\beta) = n_{eff} \cdot \frac{2\pi}{\lambda}$)
- Net optical loss (imaginary part of propagation constant)

The optical intensity is the square of the optical field, and it is normalized for use in the photon rate equation to compute the modal gain, spontaneous emission, and absorption loss (see (Eq. 15.476)–(Eq. 15.478)).

NOTE The imaginary part of the propagation constant is only given to the photon rate equation if the keyword `WaveguideLoss` is specified in the `Laser` statement.

26.4.1 Lasing wavelength in Fabry–Perot cavity

In a Fabry–Perot edge-emitting laser cavity, the wavelength is computed in the electronic solution and then passed to the Helmholtz equation solver. The physics is briefly outlined here. A Fabry–Perot cavity has eigenfrequencies:

$$\omega_p = \frac{\pi c}{\sqrt{\varepsilon_r} \cdot L} p \tag{15.484}$$

where p is an integer containing the number of wavelengths that fit into the longitudinal cavity and L is the cavity length. In a Fabry–Perot cavity (typically a few millimeters), p is usually a very large number and the longitudinal mode spacing is very narrow (a few nanometers). The modal gain has a bandwidth of the order of 10 nm. Therefore, the lasing wavelength is computed as the cavity mode that has maximum modal gain $G(\omega)$ by varying p. Changing the wavelength changes the radiative recombination in the continuity equations. The default setting updates the lasing wavelength after each Newton iteration. This is a good approximation as long as the coupling between the electronic properties and the choice of wavelength is weak. Alternatively, a self-consistent coupling is possible by including the keyword `Wavelength` in the `Plugin` statement:

```
Solve {...
   quasistationary (...
      Goal {name="p_Contact" voltage=1.6})
   {
      Plugin(BreakOnFailure){
         Coupled {Electron Hole Poisson QWeScatter QWhScatter
               PhotonRate}
         Wavelength
      }
   }
}
```

26.4.2 Specifying a fixed optical confinement factor

The Helmholtz equation is not solved if the keyword `optconfin=<float>` is specified:

```
Physics {...
   Laser (...
      Optics (...
         optconfin = 0.9
      )
   )
}
```

In this case, the spatial optical intensity Ψ is assumed to be constant over the cavity, that is, the local mode gain and spontaneous emission at each active vertex are multiplied by the constant `optconfin=<float>` factor.

26.4.3 Output power

The output power at the facet 0 of the Fabry–Perot cavity is obtained by integrating the power flow (Poynting vector) in the z-direction over the facet surface [134]. This gives:

$$P_0 = S\frac{\hbar\omega c}{2\sqrt{\varepsilon_r}L}\ln\left(\frac{1}{r_0 r_1}\right)\frac{\sqrt{r_1}(1-r_0)}{(\sqrt{r_0}+\sqrt{r_1})(1-\sqrt{r_0 r_1})} \tag{15.485}$$

The power output from the two facets is given in the current file at the end of the simulation. The keywords for specifying the cavity length and facet power reflectivities are in Table 15.140 on page 15.390.

26.5 Cavity optical modes in VCSELs

The cavity eigenproblem deals with resonance in a cavity and is a difficult problem to handle for an arbitrary geometry and inhomogeneous cavity. Cavity resonance is defined as the condition whereby a wave can sustain itself in harmony inside the cavity. This means that even after undergoing multiple internal reflections inside the cavity, the optical waves at every point in the cavity are in phase with each other. If the cavity is leaky, some waves leak and the resonance decreases in amplitude and eventually diminishes. However, in a laser cavity, when stimulated emission of photons into the resonant mode is greater than the leakage, the resonance is sustained.

The cavity eigenproblem is then a statement of how to find these resonant modes (resonant wavelength) and the required gain within the active region that will balance the leakage (optical loss) to sustain resonance, that is, laser action. Therefore, the cavity eigenproblem is different from the waveguide problem. In summary, the input is:

- Refractive index profile
- Boundary conditions

The required output for the cavity eigenproblem is:

- Resonant wavelength
- Resonant optical field profile
- Net optical loss

The detailed treatment of the cavity vectorial eigensolver for VCSELs in DESSIS has been published [185] and only a summary of key ideas is presented here. The treatment essentially follows the original idea of Henry [134], which has been extended to a nonadiabatic form [186]. First, there is the time-dependent vector wave equation:

$$\nabla \times (\nabla \times \boldsymbol{E}(\boldsymbol{r}, t)) + \frac{1}{c^2}\frac{\partial^2}{\partial t^2}[\varepsilon_r(\boldsymbol{r}, t, \tau) \otimes \boldsymbol{E}(\boldsymbol{r}, t-\tau)] = -\mu_0 \frac{\partial^2}{\partial t^2}\boldsymbol{K}(\boldsymbol{r}, t) \tag{15.486}$$

where $\boldsymbol{K}(\boldsymbol{r}, t)$ is a source term caused by spontaneous emission that contributes to the electric field density. The term in the brackets is a time convolution of the dielectric function with the electric field. The electric field is expanded (spectrally decomposed) into a discrete set of orthogonal modes,

$$\boldsymbol{E}(\boldsymbol{r}, t) = \sum_{v} a_v(t) e^{\int_0^t \omega'_v(\tau)d\tau} \Psi_v(\boldsymbol{r}, \omega) + cc \tag{15.487}$$

where $\Psi_v(\boldsymbol{r}, \omega)$ are vectorial modes, $a_v(t)$ are complex values, and cc are the complex conjugate terms. ω'_v is the frequency of the mode, a real value function. By substituting (Eq. 15.487) into (Eq. 15.486) and taking only the first-order terms of the time derivatives, a set of inhomogeneous equations is obtained:

$$\sum_{v} e^{\int_0^t \omega'_v(\tau)d\tau} \cdot \left[\left(\nabla \times (\nabla \times \Psi_v) - \frac{\omega'^2_v}{c^2}\varepsilon_r \Psi_v\right) \cdot a_v(t) + \tilde{\varepsilon}_r \Psi_v \cdot \dot{a}_v(t)\right] = -\mu_0 \frac{\partial^2}{\partial t^2}\boldsymbol{K}(\boldsymbol{r}, t) \tag{15.488}$$

where:

$$\tilde{\varepsilon}_r = \frac{2i\omega'_v}{c^2}\left(\frac{\omega'_v}{2}\frac{\partial}{\partial \omega'_v}\varepsilon_r + \varepsilon_r\right) \tag{15.489}$$

In this case, the frequency dispersion of the dielectric function has been taken into account in (Eq. 15.489). To solve the above set of inhomogeneous equations, the auxiliary homogeneous equation is introduced:

$$\left[\nabla \times (\nabla \times \quad) - \frac{\omega'^2_v}{c^2}\varepsilon_r\right] \cdot \Psi_v = \omega_v'' \tilde{\varepsilon}_r \Psi_v \tag{15.490}$$

where ω_v'' is defined to be the eigenvalue and the orthogonality relation is:

$$\int_{volume} \Psi_u \cdot \tilde{\varepsilon}_r \Psi_v d\boldsymbol{r} = \delta_{uv} \tag{15.491}$$

At resonance, ω_v'' must be purely real [185] and this forms the basis for finding the solution to the cavity eigenproblem. The frequency ω'_v is varied and (Eq. 15.490) is solved repeatedly for ω_v''. As ω'_v approaches the resonance value, the imaginary part of ω_v'' approaches zero in an approximately linear manner. This gives users an advantage in accelerating the solution-hunting. Therefore, it is important that a 'close enough' target eigenvalue is provided to benefit from this advantage.

Next, to derive the photon rate equation for each cavity mode v, (Eq. 15.490) is substituted into (Eq. 15.488) and the orthogonality relation, (Eq. 15.491), is applied. After some manipulation, the cavity photon rate equation evaluates to the familiar form:

$$\frac{d}{dt}S_v = -2\omega_v''S_v + R_v^{sp} \tag{15.492}$$

where R_v^{sp} is the spontaneous emission rate. It transpires that the eigenvalue of (Eq. 15.490), $2\omega_v''$, is the net loss rate. The material gain and absorption enter the photon rate equation indirectly through the dielectric function of the active region:

$$\varepsilon_{r,\,active}(\boldsymbol{r}, \omega'_v) = \left(n(\boldsymbol{r}, \omega'_v) - \frac{c^2}{4\omega'^2_v} r^{st}(\boldsymbol{r}, h\omega'_v)\right)^2 + i\frac{n(\boldsymbol{r}, \omega'_v)c}{\omega'_v} r^{st}(\boldsymbol{r}, h\omega'_v) \tag{15.493}$$

$r^{st}(\boldsymbol{r}, h\omega'_v)$ is the stimulated emission coefficient and is discussed in Section 28.3 on page 15.443. This approach ensures that the material gain and absorption is coupled rigorously between the electronics and optics. Therefore, the total cavity loss (or gain) is computed accurately. With the capability to model rapid changes in material gain, this cavity solver allows users to handle gain-guided VCSELs as well.

The photon rate equation is coupled to the Poisson, carrier continuity, and temperature (and hydrodynamic) equations using the Newton method, so the derivatives of $2\omega_v''$ with respect to a host of quantities for the Jacobian matrix entries are required.

$2\omega_v''$ can be identified as the relative change of the average electromagnetic energy stored in mode v:

$$2\omega_v'' = \frac{1}{\int_{volume} \langle W_v \rangle d\boldsymbol{r}} \cdot \int_{volume} \langle \frac{\partial W_v}{\partial t} \rangle d\boldsymbol{r} \tag{15.494}$$

where W_v is the energy density of the electromagnetic field of mode v derived from the Poynting vector. Assuming that the optical mode does not change greatly, the derivatives of $2\omega_v''$ can be computed from the derivatives of the dielectric function, $\varepsilon_r(\boldsymbol{r}, t, \tau)$.

26.5.1 VCSEL output power

When the vectorial optical solver is used, perfectly matched layers (PMLs) are used to surround the simulation structure. The PMLs act as wave absorbers to prevent reflections and artificially simulate the radiative boundary condition (see Section 27.5 on page 15.410). The output power emitted from the top surface is the time averaged dissipation of the optical power in the top PML.

The emitted power for mode v is, therefore, computed by the integral:

$$P_{TopEmit,\,v} = \int\limits_{vol-TopPML} \left\langle \frac{\partial W_v}{\partial t} \right\rangle d\boldsymbol{r} \qquad (15.495)$$

and takes into account the material absorption of the top PML.

26.5.2 Cylindrical symmetry

Discretizing the VCSEL structure in 3D results in a prohibitively huge mesh size. Therefore, cylindrical geometry is assumed to reduce the size of the problem. By considering cylindrical symmetry in a body-of-revolution (BOR) about the symmetry axis, the cavity modes of the VCSEL can be further decomposed into cylindrical harmonics:

$$\Psi_v(\rho, \phi, z) = \sum_m \Psi_v^{(m)}(\rho, z) \cdot e^{im\phi} \qquad (15.496)$$

It is well known that the cylindrical harmonics are orthogonal and, hence, the vectorial resonant optical field $\Psi_v^{(m)}(\rho, z)$ is solved for each cylindrical order, m, in 2D space. In the DESSIS command file, the cylindrical harmonic order m is input using the keyword `AzimuthalExpansion`. (More about the syntax is discussed in Section 27.3.1 on page 15.401.)

Using experience from optical fiber modeling, the fundamental mode of an optical fiber is HE11, followed by its immediate higher order modes, TE01, TM01, and so on. This means that the fundamental mode has cylindrical harmonic order of $m = 1$, and the next two higher order modes have orders $m = 0$. Users are encouraged to use different cylindrical harmonic orders to verify which one contains the fundamental resonant mode of the VCSEL cavity.

26.5.3 Approximate methods for VCSEL cavity problem

Apart from the rigorous vectorial treatment of VCSEL cavity optics, DESSIS also contains two approximate methods to compute the resonant modes in VCSEL cavities:

- Transfer matrix method for multilayers with a Gaussian transverse mode profile
- Effective index method

Both methods are scalar in nature and fast, and have low memory requirements. In particular, the transfer matrix method is suitable for users who want to look at how the transverse mode shape affects the different lasing characteristics of a VCSEL. The effective index method is best suited for index-guided VCSELs and it has been shown to compute accurate resonant wavelengths for many index-guided VCSEL structures including oxide-confined ones. When these approximate methods are used, DESSIS uses the same photon rate equation as for edge-emitting lasers.

26.6 Modeling light-emitting diodes

A light-emitting diode (LED) and a laser share the same physics of carrier transport. Therefore, the theory presented for quantum-well modeling in Chapter 28 on page 15.439 is applicable to LED simulations as well. The key difference between an LED and a laser is a resonant cavity design for lasers that enhances the coherent stimulated emission at a single frequency (for each mode). An LED emits with a spectrum of wavelengths based on spontaneous emission of photons in the active region. However, a new design for LEDs with a resonant cavity – the resonant cavity LED (RCLED) – uses the resonance characteristics to cause an amplified spontaneous emission in a narrower spectrum to allow for superbright emissions.

The simulation of LEDs presents many challenges. The large dimension of typical LED structures, in the range of millimeters, forbids the use of standard time-domain electromagnetic methods such as finite difference and finite element. These methods require at least 10 points per wavelength and typical emissions are at 1 μm. A quick estimate gives a necessary mesh size in the order of 10 million mesh points for a 2D geometry. Alternatively, the use of the raytracing method approximates the optical intensity inside the device as well as the amount of light that can be extracted from the device. In many cases, a 2D simulation is not sufficient and a 3D simulation is required to give an accurate account of the physical effects associated with the geometric design of the LED.

Innovative designs such as inverted pyramid structures, chamfering of various corners, and drilling holes are performed in an attempt to extract the maximum amount of light from the device. The ISE device editor DEVISE is well equipped to create complex 3D devices and provides great versatility in exploring different realistic LED designs.

26.6.1 Coupling between electronics and optics in an LED simulation

Similar to a laser simulation, an LED simulation solves the Poisson equation, carrier continuity equations, temperature equation, and Schrödinger equation self-consistently. The flowchart in Figure 15.79 illustrates the coupling of the various equation systems in an LED simulation.

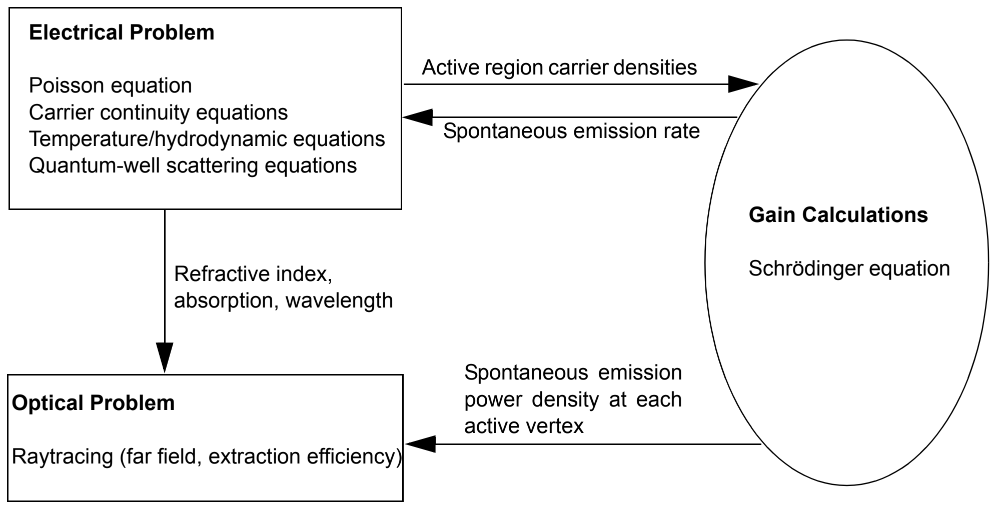

Electrical Problem
Poisson equation
Carrier continuity equations
Temperature/hydrodynamic equations
Quantum-well scattering equations
Active region carrier densities
Spontaneous emission rate
Gain Calculations
Schrödinger equation
Refractive index, absorption, wavelength
Spontaneous emission power density at each active vertex
Optical Problem
Raytracing (far field, extraction efficiency)

Figure 15.79 Flowchart of the coupling between the electronics and optics for an LED simulation

26.6.2 Discussion of LED physics

Many physical effects are manifested in an LED structure. Current spreading is important to ensure that the current is channelled to supply the spontaneous emission sources at strategic locations that will provide the optimal extraction efficiency. When quantum wells are involved, DESSIS computes a net carrier capture into the quantum wells based on scattering processes. In some cases, the LED structure gives a preferred polarization in the optical field and the spontaneous emissions in the active region can become anisotropic. DESSIS has a new feature to allow users to select the shape of the anisotropy for spontaneous emission.

The geometric shape of the LED is changed to extract more light from the structure. In most cases, the major part of the light produced is trapped within the structure through total internal reflection. As a result, the photon recycling effect becomes important. The photon recycling effect contains two parts: amplified spontaneous emission (ASE) and absorption re-emission processes. A new physical model for photon recycling has been formulated and will be introduced in DESSIS in ISE TCAD Release 11.0. Another area under development is a physical model for RCLEDs.

Important aspects of LED design are easily simulated by DESSIS. These include current spreading flow, geometric design, physics of quantum well transport, and extraction efficiency. A new feature allows users to look at the wavelength spectrum of the far-zone radiation. This is especially useful in the design of white LEDs.

CHAPTER 27 Optics

27.1 Overview

In a laser simulation, the optics problem is challenging. The refractive index distribution in a laser structure changes with temperature and carrier density (plasma effect). In DESSIS, the refractive index is taken from a corresponding parameter file and wavelength dispersion of the refractive indices can also be taken into consideration (see Section 29.5 on page 15.477). In order to handle such inhomogeneous refractive index geometries, the finite element method was selected as the core method to discretize and solve the optics problem. In addition, there are other approximate methods such as the effective index method, transfer matrix method, and raytracing. These additional methods allow users to run the simulation faster, but with some loss of accuracy. They are described in detail in the following sections.

The resulting quantities of interest from the optics solution are the optical mode profiles, optical loss, and resonant wavelength. In a Fabry–Perot cavity for an edge-emitting laser, the wavelength is determined by the peak of the material gain from the electrical simulation. In other cases such as DFB lasers, DBR lasers, and VCSELs, the wavelength is determined by the shape and design of the cavity structure.

The optics problem can be iterated self-consistently with the electrical problem by Gummel iteration. Referring to Section 25.2.1 on page 15.373, the self-consistent coupling is activated by using the keyword `Optics` in the `Plugin` statement:

```
Solve {...
   quasistationary (...
      Goal {name="p_Contact" voltage=1.6})
   {
      Plugin(BreakOnFailure){
         Coupled { Electron Hole Poisson QWeScatter QWhScatter
                      PhotonRate }
         Optics
         Wavelength
      }
   }
}
```

If the coupling between the optics and electronics is weak, for example, in an isothermal simulation and at low-current injection conditions, it is not necessary to iterate the optics problem with the electrical problem since the optical eigenmode is not expected to change greatly. In such a case, remove the `Optics` keyword from the `Plugin` statement, and the optical eigenmode is computed only once at the beginning of the simulation.

There are two types of optical problem in laser simulations: the waveguide problem for edge-emitting lasers and the resonant cavity problem, for example, in VCSELs. In the waveguide problem, essentially, the Helmholtz equation is solved, while the cavity problem solves the wave equation directly. These two types of problem are discussed in the next sections.

27.2 Finite element (FE) formulation

The finite element method is a standard variational approach for solving the electromagnetic wave equations [187]. The Ritz procedure is used to evaluate the functional:

$$F(\Psi) = \frac{1}{2}\langle \Im\Psi, \Psi\rangle - \langle \Psi, f\rangle - \langle f, \Psi\rangle \quad (15.497)$$

where $\Im$ is the linear operator and Ψ is a trial function. The governing differential equation is $\Im\Phi = f$. The wave equations in the waveguide and cavity problems are both of the form of the governing differential equation and, therefore, are well suited to the finite element method.

For example, use the scalar Helmholtz equation for the waveguide problem to trace the formulation of the finite element method. The Ritz procedure is applied to the scalar Helmholtz equation and gives the variational functional:

$$F(\Psi) = \frac{1}{2}\iint((\partial_x\Psi)^2 + (\partial_y\Psi)^2 - k_0^2\varepsilon_r\Psi^2)d\Omega \quad (15.498)$$

The edge-emitting laser structure is discretized into finite elements, and the trial function Ψ is constructed by assuming weighted, linear basis functions on each edge of the finite element. These basis functions are commonly referred to as shape functions. The variables, in this case, are the coefficients (weights) of the linear basis function on each edge. By the method of variation, the minimization of the functional $F(\Psi)$ gives the optimal solution, Ψ, for the finite element discretization of the scalar Helmholtz equation. Therefore, the resolution of the discretization is important to determine the accuracy of the solution. As a general rule, 20 points per wavelength will provide an accurate solution for most structures.

To find the minimization of the functional $F(\Psi)$, take the first derivative of $F(\Psi)$ with respect to the weights of the linear basis function, $\{\Phi\}$, and set the derivative to zero. This yields a set of linear equations that can be cast into an algebraic, generalized, eigenvalue problem:

$$[A]\{\Phi\} = \varepsilon_{eff}[B]\{\Phi\} \quad (15.499)$$

with:

$$[A] = \sum_{allElements}\left(k_0^2\varepsilon_{r,e}\iint_{\Omega^e}\{L\}\cdot\{L\}^T d\Omega^e - \iint_{\Omega^e}\{\nabla_t L\}\cdot\{\nabla_t L\}^T d\Omega^e\right) \quad (15.500)$$

$$[B] = \sum_{allElements}\iint_{\Omega^e}\{L\}\cdot\{L\}^T d\Omega^e \quad (15.501)$$

where $\{L\}$ are the linear shape functions. The relative dielectric constant ε_r remains constant across the element e, and the integration occurs over the area of the element, Ω^e.

The sparse matrices $[A]$ and $[B]$ are derived from the assembly process. They are complex symmetric but nonhermitian. As the system is nonhermitian, it causes the eigenvalues ε_{eff} to be complex. $\{\Phi\}$ is the column vector of the unknown weights. After the weights, $\{\Phi\}$, have been solved, they are substituted back into the linear shape functions to recover the electric fields on the nodes of the grid.

The Jacobi–Davidson QZ algorithm is applied [136][137] to solve the sparse matrix generalized eigenvalue problem. This is an iterative solver, so an initial guess for the eigenvalue is required. The better the initial guess, the faster the solution will be found. In addition, the numeric solver has also been parallelized.

27.3 Syntax of FE scalar and FE vectorial optical solvers

The finite element (FE) scalar solver caters only to the waveguide problem, while the FE vectorial solver handles both waveguide and cavity problems. The choice of FE scalar or FE vectorial solvers is determined in the `Optics` statement. The default is the FE scalar solver for waveguide modes.

27.3.1 FE scalar solver

The `FEscalar` solver for the waveguide problem is activated in the `Physics-Laser-Optics` statement in the command file:

```
Physics {...
   Laser (...
      Optics(
         FEScalar(EquationType = Waveguide
               Symmetry = Symmetric
               LasingWavelength = 800         # [nm]
               TargetEffectiveIndex = 3.4     # initial guess
               TargetLoss = 10.0              # initial guess [1/cm]
               Polarization = TE
               Boundary = "Type1"
               ModeNumber = 1
         )
      )
   )
}
```

This example shows how to activate the FE scalar solver to solve for one waveguide mode. As discussed in Section 27.2 on page 15.400, an iterative method is used to solve for the modes. Therefore, specifying a good `TargetEffectiveIndex` is important to speed up the computation of the solution. For multimodes, increase `ModeNumber` and specify multiple entries for `TargetEffectiveIndex`, `TargetLoss`, and so on (see Section 27.3.3 on page 15.405). Only Cartesian coordinates and waveguide modes are associated with the FE scalar solver. For scalar cavity solvers for VCSELs, the user is referred to the transfer matrix method and effective index method.

In DESSIS, there is an option to run only the optical solver without activating the laser simulation. This is called the optics stand-alone option (see Section 29.8 on page 15.484). In this case, users must specify the keyword `Boundary` in the `FEScalar` statement if `Symmetric` or `Periodic` is chosen. In a laser simulation, the default values of `Boundary` listed in Table 15.147 on page 15.407 are used unless users select the boundary types explicitly. A detailed discussion of the `Boundary` keyword is in Section 27.4 on page 15.406.

Table 15.141 and Table 15.142 describe all of the possible arguments that can be used inside the `FEScalar` statement. A brief explanation of some keywords follows.

Table 15.141 Arguments for the FEScalar statement

Feature keyword	Parameter keyword/Description	Default value
`EquationType=<parameter>`	`Waveguide`	`Waveguide`

`Symmetry=<parameter>`	`Symmetric`
	`NonSymmetric`
	`Periodic`
		`Symmetric`

`LasingWavelength=<float>`	Lasing wavelength [nm]	–1.0
`TargetEffectiveIndex=<float>`	Initial guess for effective refractive index	–1.0
`TargetLoss=<float>`	Initial guess for propagation loss [1/cm]	0.0
`Polarization=<parameter>`	`TE` `TM`	`TE`
`ModeNumber=<int>`	Number of modes to solve	1
`Boundary="<parameter>"`	`Type1` `Type2`	No default
`Absorption(ODB)`	Allows user to specify the material loss in the optical database section of the material parameter file	No default

Table 15.142 Dependences of keywords for FEScalar optical mode solver

Keyword	Dependencies				
`EquationType`	`Waveguide`				
`Coordinates`	Not used; Cartesian is always assumed for `FEScalar`.				
`Symmetry`	`Symmetric`		`NonSymmetric`	`Periodic`	
`Boundary`	`Type1`	`Type2`	Not valid	`Type1`	`Type2`
`LasingWavelength`	`<float>`				
`TargetEffectiveIndex`	`<float>`				
`TargetLoss`	`<float>`				

There are three types of symmetry entry: `Symmetric` is symmetry about the y-axis at x = 0, `Nonsymmetric` is nonsymmetry, and `Periodic` means the left and right boundaries of the Neumann type. The default boundary conditions for the different symmetry types are listed in Table 15.147 on page 15.407. The `LasingWavelength` entered is solely for the initial computation of the optical mode and the way the lasing wavelength is computed is discussed in Section 26.4 on page 15.390. `TargetEffectiveIndex` is the initial guess for the eigenvalue of the waveguide problem and is a necessary input to ensure that the required mode is computed.

27.3.2 FE vectorial solver

The FE vectorial solver handles both waveguide and cavity-type problems. The syntax of both problems is described separately.

27.3.2.1 FE vectorial solver for waveguides

The syntax for using the vectorial FE solver for waveguides is:

```
Physics {...
   Laser (...
      Optics(
         FEVectorial(EquationType = Waveguide
                  Symmetry = Symmetric
                  Coordinates = Cartesian
                  LasingWavelength = 800          # [nm]
                  TargetEffectiveIndex = 3.4      # initial guess
                  TargetLoss = 10.0               # initial guess [1/cm]
                  Boundary = "Type1"
                  ModeNumber = 1
        )
      )
    )
}
```

The argument list for `FEVectorial` statement for the waveguide problem is similar to that of the `FEScalar` statement.

27.3.2.2 FE vectorial solver for VCSEL cavity

The syntax for using the FE vectorial solver for a VCSEL cavity is:

```
Physics {...
   Laser (...
      Optics(
         FEVectorial(EquationType = Cavity
                  Coordinates = Cylindrical
                  TargetWavelength = 777       # initial guess [nm]
                  TargetLifeTime = 1.4         # initial guess [ps]
                  AzimuthalExpansion = 1       # cylindrical harmonic order
                  ModeNumber = 1
         )
      )
      VCSEL()          # specify this is a VCSEL simulation
   )
}
```

The syntax for the VCSEL cavity problem is very different from that of the waveguide problem. The keyword `VCSEL` must be included in the `Laser` statement to indicate that this is a VCSEL simulation. An initial guess for the resonant wavelength must be specified by `TargetWavelength`.

Cylindrical symmetry has been specified by `Coordinates=Cylindrical`, so the modes contain the angular dependence of $e^{im\phi}$. The cylindrical harmonic order, m, can be changed by the keyword `AzimuthalExpansion`.

NOTE Unless stated specifically, the argument should apply to both cavity and waveguide problems.

Table 15.143 describes the arguments that can be used with the `FEVectorial` keyword. Table 15.144 and Table 15.145 on page 15.405 group the keywords for the waveguide and cavity problems, respectively.

Table 15.143 Arguments for the FEVectorial statement

Feature keyword	Parameter keyword/Description	Default value
`EquationType=<parameter>`	`Cavity` Solves the cavity problem	No default
	`Waveguide` Solves the waveguide problem	
`Symmetry=<parameter>`	`Symmetric` `Periodic` `NonSymmetric`	`Symmetric`
`Boundary="<parameter>"`	`Type1` `Type2`	No default
`LasingWavelength=<float>`	Lasing wavelength [nm] (for `Waveguide` only)	–1.0
`TargetEffectiveIndex=<float>`	Initial guess for effective refractive index (for `Waveguide` problems only)	–1.0
`TargetLoss=<float>`	Initial guess for propagation loss [1/cm] (for `Waveguide` problems only)	0.0
`Coordinates=<parameter>` (for cavity problems only)	`Cartesian` `Cylindrical`	No default
`TargetWavelength=<float>`	Target value for lasing wavelength [nm] (for cavity problems only)	–1.0
`TargetLifetime=<float>`	Target value for photon lifetime [ps] (for cavity problems only)	1e99
`LongitudinalWavevector=<float>`	Longitudinal wavevector (= $2\pi/\lambda_z$) [1/m] (for Cartesian cavity problems only)	–1.0
`AzimuthalExpansion=<int>`	Cylindrical harmonic order, m in $e^{im\phi}$ (for cylindrical cavity problems only)	–1.0
`Modenumber=<int>`	Number of modes to be calculated	1
`Absorption(ODB)`	Allows users to specify material loss in the optical database section of the material parameter file	No default

Table 15.144 Dependencies of keywords for EquationType=Waveguide in FEVectorial optical solver

Keyword	Dependencies				
`EquationType`	`Waveguide`				
`Coordinates`	Not valid				
`Symmetry`	`Symmetric`		`NonSymmetric`	`Periodic`	
`Boundary`	`Type1`	`Type2`	Not valid	`Type1`	`Type2`
`TargetWavelength`	Not valid				
`TargetLifetime`	Not valid				

Table 15.144 Dependencies of keywords for EquationType=Waveguide in FEVectorial optical solver

Keyword	Dependencies
`LongitudinalWavevector`	Not valid
`AzimuthalExpansion`	Not valid
`LasingWavelength`	`<float>`
`TargetEffectiveIndex`	`<float>`
`TargetLoss`	`<float>`

Table 15.145 Dependencies of keywords for EquationType=Cavity in FEVectorial optical solver

Keyword	Dependencies			
`EquationType`	`Cavity`			
`Coordinates`	`Cartesian`			`Cylindrical`
`Symmetry`	`Symmetric`		`NonSymmetric`	Not valid
`Boundary`	`Type1`	`Type2`	Not valid	
`TargetWavelength`	`<float>`			
`TargetLifetime`	`<float>`			
`LongitudinalWavevector`	`<float>`			Not valid
`AzimuthalExpansion`	Not valid			`<int>`
`LasingWavelength`	Not valid			
`TargetEffectiveIndex`	Not valid			
`TargetLoss`	Not valid			

27.3.3 Specifying multiple entries for parameters in FEScalar and FEVectorial

When multimodes are needed, `ModeNumber` is used to specify the number of modes required. In this case, users may want to specify different initial guesses for the effective index (for waveguide modes), resonant wavelength (for cavity modes), and so on. This is accomplished by extending the syntax for these initial guess parameters. The following is an example for waveguide modes and one for VCSEL cavity modes.

27.3.3.1 Multiple waveguide modes

The syntax extension of `FEScalar` for multiple waveguide modes is shown here. `FEVectorial` for multiple waveguide modes has the same syntax extension. The boundary type is specified explicitly in this example. If the `Boundary` keyword is omitted, the default values in Table 15.147 on page 15.407 are taken:

```
Physics {...
   Laser (...
      Optics(
         FEScalar(EquationType = Waveguide
               Symmetry = Symmetric
```

```
                LasingWavelength = 780
                TargetEffectiveIndex = (3.54 3.47 3.5 3.33 3.4)
                TargetLoss = (5.0 6.0 8.0 7.0 10.0)
                Polarization = (TE TE TM TE TM)
                ModeNumber = 5
                Boundary = ("Type1" "Type2" "Type1" "Type1" "Type2")
            )
         )
      )
   }
```

27.3.3.2 Multiple cavity modes

The syntax extension of `FEVectorial` for multiple cavity modes is:

```
Physics {...
   Laser (...
      Optics(
         FEVectorial(EquationType = Cavity
                     Coordinates = Cylindrical
                     TargetWavelength = (777 762 760 753 751)
                     TargetLifeTime = (1.5 1.3 1.22 1.16 1.1)
                     AzimuthalExpansion = (1 0 0 2 2)
                     ModeNumber = 5
         )
      )
      VCSEL()
   )
}
```

27.4 Boundary conditions and symmetry for optical solvers

There are three main types of boundary condition for the optical problem:

- Dirichlet boundary condition
- Neumann boundary condition
- Radiative boundary condition

These boundary conditions can be controlled to some extent with the `Symmetry` keyword in the `FEScalar` and `FEVectorial` statements. In the DESSIS optical solvers, the external boundaries of the optical simulation space are assumed (by default) to satisfy the Dirichlet boundary condition, that is, the optical fields on this outer boundary are set to zero. By specifying different types of symmetry, the Neumann boundary condition can be imposed on different external boundaries.

Table 15.146 summarizes the symmetry types (appearing in the `FEScalar` and `FEVectorial` statements) and lists the associated boundary conditions.

Table 15.146 Symmetry types in FEScalar and FEVectorial statements and their associated optical boundary conditions

Symmetry type	Boundary condition
`NonSymmetric`	Dirichlet boundary condition on all external boundaries, that is, the optical field is set to zero at the boundaries.
`Periodic`	Neumann or Dirichlet boundary conditions on all vertical boundaries, and Dirichlet boundary condition on all horizontal boundaries.
`Symmetric`	Neumann or Dirichlet boundary condition for even or odd on the symmetry axis, and Dirichlet boundary condition elsewhere. The symmetry axis is defined as the y-axis at x = 0.

If `Symmetry=Symmetric` and the optics stand-alone option is used, the argument keyword `Boundary` must be specified. In laser simulations, `Boundary` is chosen automatically unless explicitly specified. The argument `Boundary` dictates the boundary condition at the symmetry y-axis and is specific only for Cartesian coordinates because odd and even modes require different types of boundary condition at the symmetry y-axis. A summary of the default boundary conditions used in different cases is presented in Table 15.147. The following examples explain the different cases.

Table 15.147 Default boundary conditions for the optical solvers

Case	Periodic	Symmetric	NonSymmetric
`FEScalar`	`Boundary = "Type1"` for first n/2 modes and `Boundary = "Type2"` for the rest of the modes.		Boundary not used.
`FEVectorial, Waveguide`	`Boundary = "Type2"` for first n/2 modes and `Boundary = "Type1"` for the rest of the modes.		
`FEVectorial, Cavity`	Boundary chosen automatically.		

NOTE The boundary condition for every mode can be explicitly specified by the keyword `Boundary = ("Type2" "Type1" ...)`.

27.4.1 Symmetric FEScalar waveguide mode in Cartesian coordinates

In this example, the `FEScalar` optical mode solver is applied to a symmetric edge-emitting laser to obtain the even and odd modes. The top row of Figure 15.80 on page 15.408 shows the even modes. They are calculated when the keyword `Boundary = "Type1"` is set. The bottom row shows the odd modes. They are computed when the keyword `Boundary = "Type2"` is set. The fundamental mode is obtained if the keyword `Boundary = "Type1"` is set while the first-order mode is calculated for `Boundary = "Type2"`.

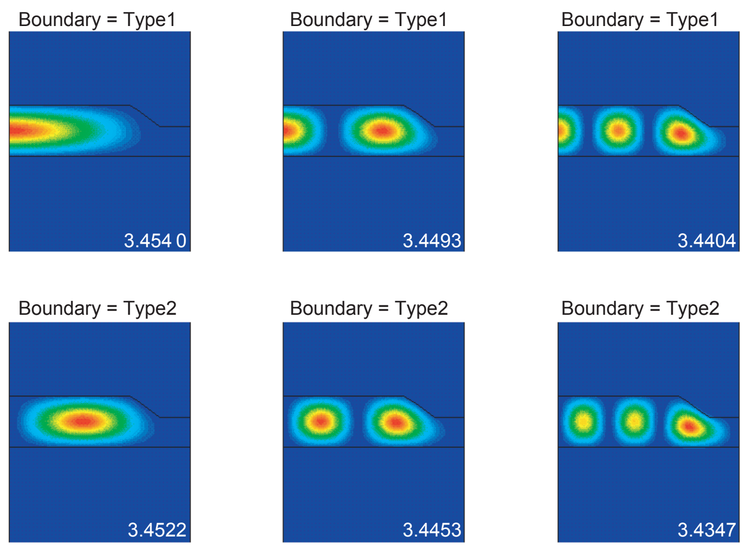


Figure 15.80 Scalar mode patterns of a symmetric edge-emitter structure

27.4.2 Symmetric FEVectorial waveguide modes in Cartesian coordinates

In this example, the `FEVectorial` optical mode solver is applied to a symmetric edge-emitting laser to find the horizontally and vertically polarized modes. The top row of Figure 15.81 shows the calculated modes when the keyword `Boundary = "Type1"` is set. The bottom row shows the modes that are calculated when the argument `Boundary = "Type2"` is set. The horizontally polarized fundamental mode is obtained for `Boundary = "Type2"`.

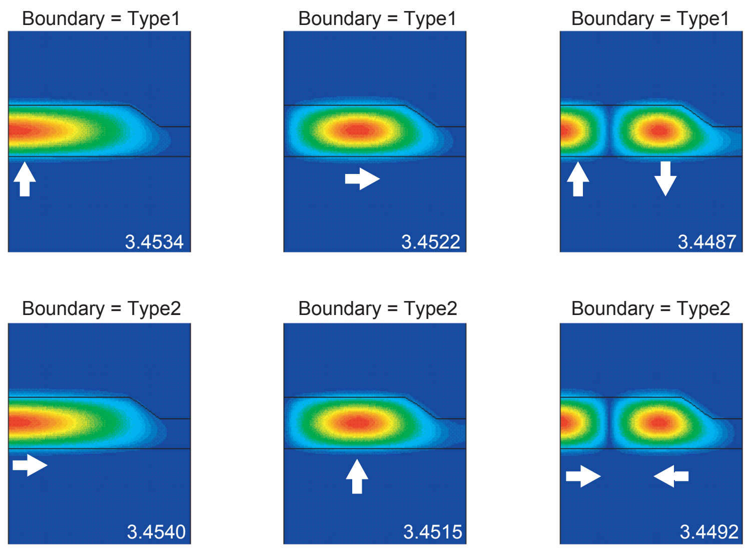


Figure 15.81 Vectorial mode patterns of a symmetric edge-emitter structure

27.4.3 Symmetric FEVectorial VCSEL cavity modes in Cartesian coordinates

In addition to cylindrical symmetry, the `FEVectorial` optical mode solver can also be applied to a symmetric VCSEL in Cartesian coordinates to find the modes that are polarized in-plane or perpendicular to the plane. The top row of Figure 15.82 shows the modes when the keyword `Boundary = "Type1"` is set. The bottom row shows the modes that are computed when the keyword `Boundary = "Type2"` is set. The in-plane polarized fundamental mode is obtained for `Boundary = "Type2"`, while the perpendicularly polarized fundamental mode is computed for `Boundary = "Type1"`.

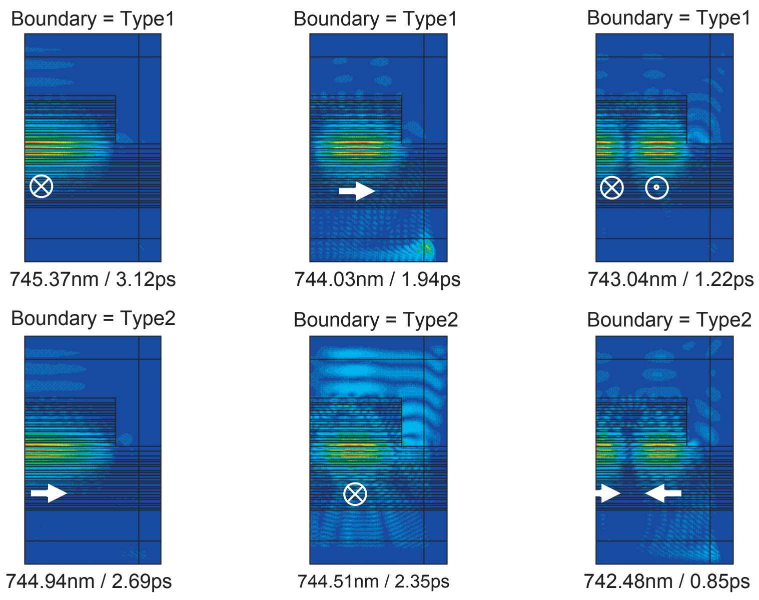


Figure 15.82 Vectorial mode patterns of a VCSEL in Cartesian coordinates

27.4.4 Symmetric FEVectorial VCSEL cavity modes in cylindrical coordinates

The argument `AzimuthalExpansion` in the `FEVectorial` optical mode solver is used to calculate different types of mode in a cylindrical VCSEL. The left column of Figure 15.83 on page 15.410 shows the computed modes for the argument `AzimuthalExpansion=0`. The middle column shows the modes that are calculated when `AzimuthalExpansion=1` is set. The right column shows the mode pattern for `AzimuthalExpansion=2`. For example, the fundamental mode HE11 is obtained by setting the argument `AzimuthalExpansion=1`.

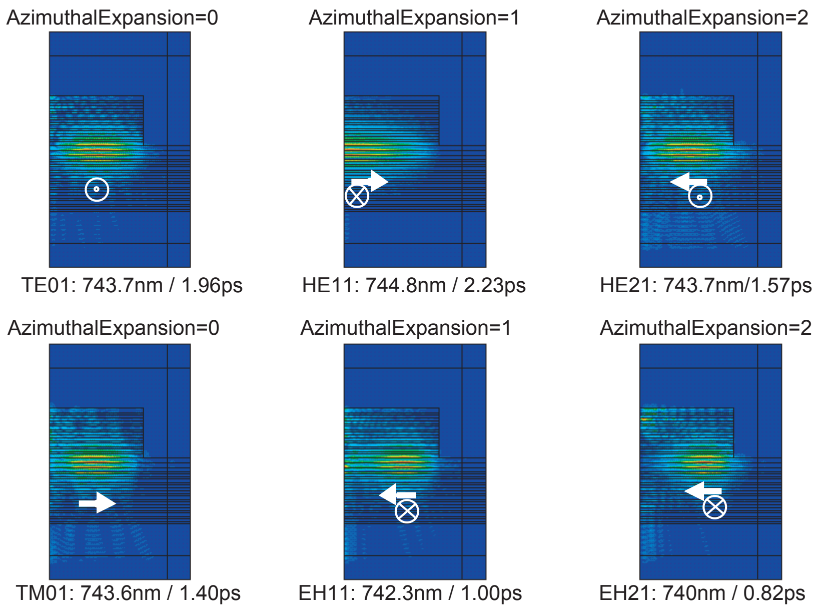


Figure 15.83 Vectorial mode patterns of a VCSEL in cylindrical coordinates

27.5 Perfectly matched layers

Apart from Dirichlet and Neumann boundary conditions, DESSIS–Laser can simulate the radiative boundary condition artificially using the concept of a perfectly matched layer (PML).

The PML in DESSIS is implemented by using a tensorial permeability quantity [185], which can be interpreted as a uniaxial anisotropic medium. This can be proven to be the same as coordinate stretching for the curl, divergence, and gradient operators [189]. (Further information is available from the literature: the original PML work [188], PML interpreted as a coordinate-stretching concept [189], and the generalization of the PML concept to various coordinate systems and general anisotropic and dispersive media [190][191].) A mathematical presentation of the PML is beyond the scope of this manual. Therefore, only the physical concept behind the method is discussed.

NOTE Perfectly matched layers should only be added when using the vectorial (`FEVectorial`) optical solver.

The Dirichlet boundary condition is assumed at the outer boundaries of the simulation space. In many cases, the optical field may radiative outwards, for example, waves from the lasing region leak into the substrate because the substrate has about the same refractive index as the guiding layers. In this case, the solution obtained using the Dirichlet boundary condition will include reflections of these radiative waves from the boundaries, which are nonphysical. To solve this problem, the boundaries of the structure can be coated with PMLs to reduce and eliminate unwanted reflections from the Dirichlet boundary condition in a truncated simulation space. This enables the artificial simulation of the radiative boundary condition.

The structure is terminated by an inner boundary Γ_i (see Figure 15.84), and the PML is placed between the inner boundary Γ_i and outer boundary Γ_o. A gradually increasing loss is introduced in the PML from the inner boundary Γ_i towards the outer boundary Γ_o. Therefore, upon first impact of the wave into the PML at the inner boundary Γ_i, negligible reflection occurs. As the wave propagates deeper into the PML, it is absorbed more and more. Any reflected waves within the PML also suffer from absorption. Ultimately, it appears that the propagating wave in the PML is totally absorbed and, therefore, this is how the PML simulates the radiative boundary condition. The loss profile in the PML has been set automatically in DESSIS.

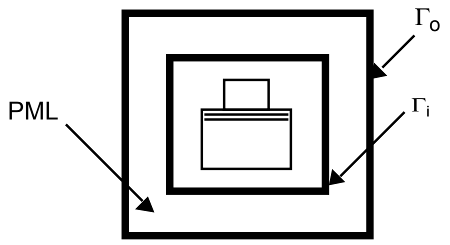


Figure 15.84 PML coating the structure

PMLs are indicated by special keywords appended to the beginning of the region names when the optical device is drawn. The region names of PMLs at the top, bottom, left, and right of the structure start with `TPML`, `BPML`, `LPML`, and `RPML`, respectively, as shown in Figure 15.85.

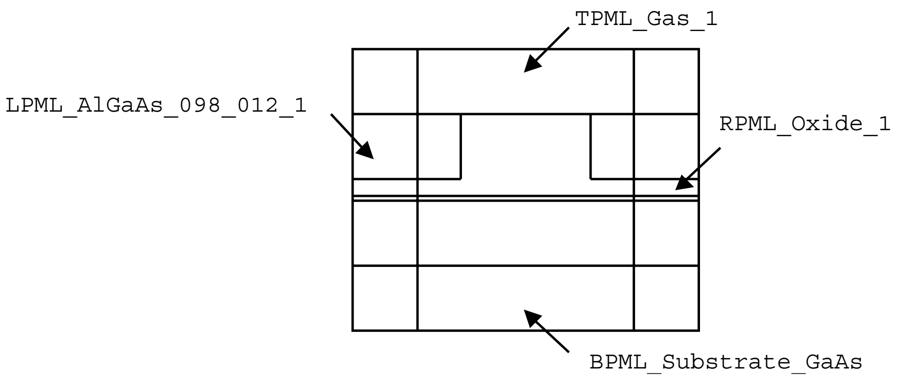


Figure 15.85 Naming of PML regions

This naming convention is required to impose the correct boundary condition in the optical solver. These special PML regions are declared in the same way as other regions in the DESSIS command file. A Tcl-based script can be used to add automatically PML regions to the outer boundaries of the structure. This script can be obtained from ISE Technical Support.

27.6 Transfer matrix method for VCSELs

The 1D transfer matrix method (TMM) is a simplified scalar solver for the VCSEL cavity problem. It computes the spatial optical intensity and the corresponding characteristics of the fundamental mode in a cylindrically symmetric VCSEL structure. The scalar wave equation in the axial direction is:

$$\left(\frac{\partial^2}{\partial z^2} + k_z^2\right)\phi_z = 0 \qquad (15.502)$$

where ϕ_z denotes the wave component in the axial direction.

The TMM is applied at the symmetry axis in the vertical direction (see Figure 15.86).

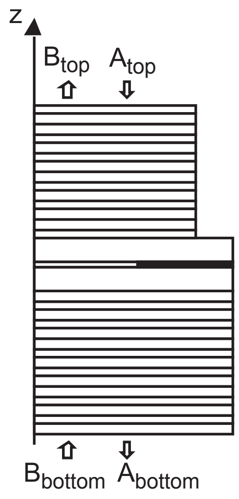


Figure 15.86 Transfer matrix method applied to a symmetric VCSEL structure

The solution to the axial wave equation in each layer *i* can be expressed as the sum of a forward-propagating and backward-propagating wave:

$$\phi_z = A(z_i)e^{(ik_{zi}z)} + B(z_i)e^{(-ik_{zi}z)} \tag{15.503}$$

The TMM relates the two waves at the interfaces *i* and *i*+1 by:

$$\begin{bmatrix} A(z_i) \\ B(z_i) \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix}_i \cdot \begin{bmatrix} A(z_{i+1}) \\ B(z_{i+1}) \end{bmatrix} \tag{15.504}$$

The transfer matrix for an index step $n_1 \rightarrow n_2$ is:

$$T_{1\rightarrow 2} = \frac{1}{t_{12}} \begin{bmatrix} 1 & r_{12} \\ r_{12} & 1 \end{bmatrix} \tag{15.505}$$

with $r_{12} = -r_{21} = \dfrac{n_1 - n_2}{n_1 + n_2}$ and $t_{12} = t_{21} = \dfrac{2n_1}{n_1 + n_2}$.

For a homogeneous medium of length *d*, the transfer matrix becomes:

$$T_d = \begin{bmatrix} e^{ikd} & 0 \\ 0 & e^{-ikd} \end{bmatrix} \tag{15.506}$$

With these basic transfer matrix blocks, a final transfer matrix can be set up that relates the forward-propagating and backward-propagating waves at the top (A_{top}, B_{top}) of the device to the wave at the bottom (A_{bottom}, B_{bottom}) of the device:

$$\begin{bmatrix} B_{top} \\ A_{top} \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix}_0 \cdot \ldots \cdot \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix}_{2n+1} \cdot \begin{bmatrix} B_{bottom} \\ A_{bottom} \end{bmatrix} = \begin{bmatrix} T_{11,\,tot} & T_{12,\,tot} \\ T_{21,\,tot} & T_{22,\,tot} \end{bmatrix} \cdot \begin{bmatrix} B_{bottom} \\ A_{bottom} \end{bmatrix} \tag{15.507}$$

where n denotes the number of layers of the structure. Resonance is achieved only if the outward-propagating waves are established at the top and bottom of the device ($A_{top}= 0$ and $B_{bottom}= 0$). This condition is found by varying the frequency ω and the gain in the quantum wells. The corresponding characteristics of this instance are the resonant wavelength and quantum well gain. At sustained resonance, the threshold gain in the quantum wells must balance the radiation losses from the device.

Since the 1D TMM only provides the change of field in the axial direction, a Gaussian variation is assumed for the optical intensity in the transverse direction:

$$Gauss(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \qquad (15.508)$$

where x is the distance in the transverse direction from the symmetry axis and σ is the width of the Gaussian shape.

The activation syntax for the TMM for VCSEL cavity problems in the command file is:

```
Physics {...
   Laser (...
      Optics(
         TMM1D(SigmaGauss=4.0           # width of Gaussian, [micron]
               TargetWavelength=800       # initial guess [nm]
         )
      )
      VCSEL()                             # specify this is a VCSEL simulation
   )
}
```

The solution of the resonant wavelength is solved iteratively, so users must specify an initial guess in `TargetWavelength`. The default and only symmetry type supported by `TMM1D` is `Symmetric`; only the fundamental mode is computed. The rest of the entries in the `Physics` and `Laser` statements are similar to that in Section 25.2.1 on page 15.373. Table 15.148 lists the arguments of `TMM1D`.

Table 15.148 Arguments for the TMM1D statement

Argument keyword	Description
`Absorption(ODB)`	Allows users to specify the material loss in the optical database section of the material parameter file.
`SigmaGauss=<float>`	Width of the Gaussian distribution [μm].
`TargetWavelength=<float>`	Target value for lasing wavelength [nm]. Only one value can be input because `TMM1D` solves the fundamental mode only.

27.7 Effective index method for VCSELs

The effective index method (EIM) is a fast scalar solver and well suited to computing fairly accurate resonant wavelength and optical intensity for index-guided VCSELs. However, the EIM cannot compute the scattering losses accurately for very small aperture (less than 2 μm radius) VCSELs. Nevertheless, the EIM is an option in DESSIS to cater to the rapid design of index-guided VCSELs, including oxide-confined VCSELs.

Figure 15.87 shows a typical VCSEL geometry suitable for the EIM. The structure is divided into two regions: the core and the cladding. Within each core and cladding region, multiple homogeneous layers of semiconductor are allowed.

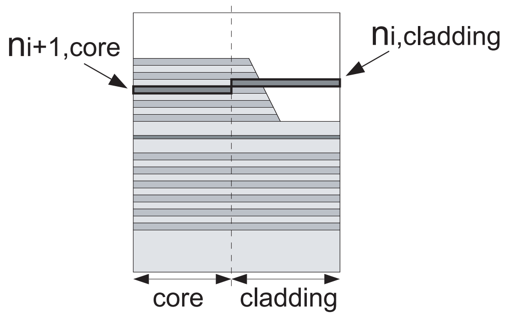


Figure 15.87 VCSEL partitioned into the core and cladding regions for the EIM

NOTE In DESSIS, the EIM is only applicable to the VCSEL cavity problem, *not* the waveguide problem.

27.7.1 Formulation of effective index method

The EIM solves the scalar Helmholtz equation:

$$\nabla^2\Psi + k^2\Psi = 0 \tag{15.509}$$

by assuming that the wavefunction Ψ is separable, that is:

$$\Psi = \phi_t(x, y) \cdot \phi_z(z) \tag{15.510}$$

where the subscripts t and z refer to the transverse and z components. Substituting (Eq. 15.510) into (Eq. 15.509) gives two independent equations: the axial wave equation:

$$\left[\frac{\partial^2}{\partial z^2} + k_z^2\right] \cdot \phi_z = 0 \tag{15.511}$$

and the transverse wave equation:

$$\left[\nabla_t^2 + \left[n(r) \cdot \frac{2\pi}{\lambda_0}\right]^2 - \beta^2\right] \cdot \phi_t = 0 \tag{15.512}$$

The transverse and axial wave equations are coupled using the dispersion relation:

$$k^2 = k_t^2 + k_z^2 = n^2 \cdot k_0^2 \tag{15.513}$$

where the free space wavenumber is $k_0 = \frac{2\pi}{\lambda_0}$.

The solution strategy of the EIM for VCSELs is best illustrated by the flowchart in Figure 15.88.

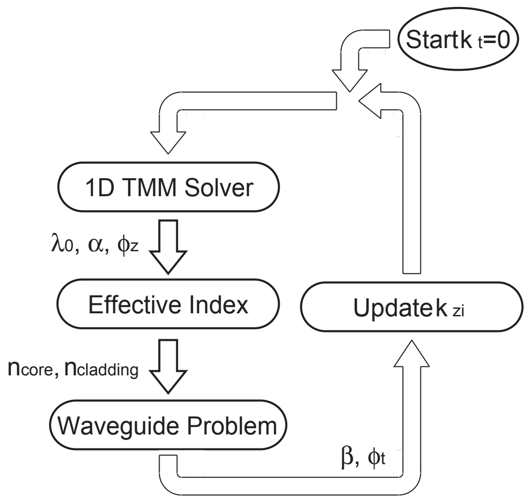


Figure 15.88 Solving the axial and transverse wave equations self-consistently in the EIM

First, the axial wave equation is solved by the transfer matrix method. The required output is the resonant wavelength λ_0, the net cavity loss α, and the z-resonant field $\phi_z(z)$.

Next, $\phi_z(z)$ is used to compute the effective indices in the core (n_{core}) and the cladding ($n_{cladding}$) regions by the following averaging relation:

$$n^2{}_{core,\,clad} = \frac{\sum\limits_i \int\limits_{z_i}^{z_{i+1}} n^2_{i,\,core/cladding} \cdot |\phi_z|^2 (dz)}{\int |\phi_z|^2 dz} \tag{15.514}$$

The refractive index of each layer i is weighted by the z-resonant optical intensity, and the effective index can be viewed as an average of these weighted refractive indices.

n_{core} and $n_{cladding}$ are subsequently used in the transverse wave equation to formulate an optical fiber or waveguide-type problem. Solving the transverse wave equation yields the propagation constant β, which is then used to update k_{z_i} of the axial wave equation using the relations:

$$\left(n_{core} \cdot \frac{2\pi}{\lambda_0}\right)^2 = k_t^2 + \beta^2 \tag{15.515}$$

and:

$$k_{z_i}^2 = \left(n_i \cdot \frac{2\pi}{\lambda_0}\right)^2 - k_t^2 \tag{15.516}$$

This is performed so that the concept of phase-matching is enforced at the tangential boundaries of all the layers. The entire process is iterated until convergence is achieved.

27.7.2 Transverse mode pattern of VCSELs

The transverse wave equation ((Eq. 15.512)) in the EIM can be solved in two different coordinate systems: Cartesian and cylindrical coordinates. The core and cladding effective indices form a symmetric three-layer waveguide problem. In Cartesian coordinates, this is equivalent to the classic step-index planar waveguide problem. In cylindrical coordinates, this becomes an optical fiber problem. Both problems can be solved by a semianalytic approach, and the range of effective index in this waveguide problem is:

$$n_{cladding} < n_{eff} < n_{core} \tag{15.517}$$

where the propagation constant $\beta = n_{eff} \cdot k_0$. This ensures that the transverse mode is a guided mode.

The step-index planar waveguide in Cartesian coordinates is an approximation to solving the transverse mode profile in square aperture VCSELs. It is assumed that the square aperture VCSEL is symmetric to a plane, as shown in Figure 15.89. The core and cladding indices form the step-index layers of the waveguide, and the TE modal field component, $\phi_t(x, y) = \phi_y(x)$, varies as $\sin(x)$ or $\cos(x)$ in the core region and as $\exp(-|x|)$ in the cladding region.

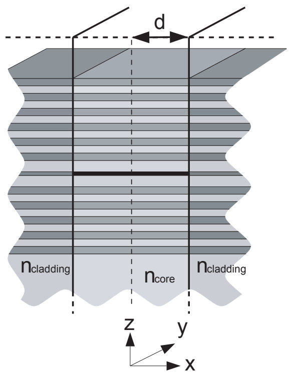


Figure 15.89 Transverse mode in square aperture VCSEL is treated as a step-index planar waveguide problem

In a circular aperture VCSEL (see Figure 15.90), the optical fiber problem is solved. The modal field components, $\phi_t(r, \varphi)$, vary as $J_{m-1}(r)$ (Bessel function) in the core region and as $K_{m-1}(r)$ (modified Bessel function) in the cladding region. The parameter m denotes the cylindrical harmonic order, $e^{im\varphi}$. The modes in an optical fiber are commonly classified as HE_{mn}, EH_{mn}, TE_{0n}, and TM_{0n}, and this convention is followed in the command file syntax. Generally, the fundamental mode is HE_{11}, followed by the higher order modes TE_{01}, TM_{01}, and so on.

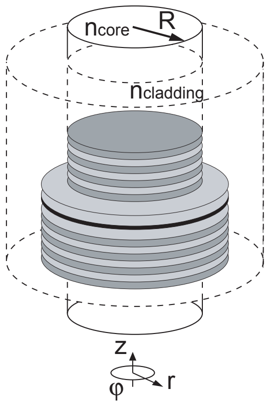


Figure 15.90 Transverse mode in circular aperture VCSEL is treated as an optical fiber problem

27.7.3 Syntax for the effective index method

The EIM solver can be activated by specifying the keyword `EffectiveIndex` in the `Physics-Laser-Optics` part of the command file.

Cylindrical modes

```
Physics {...
   Laser (...
      Optics (
         EffectiveIndex (
              TargetWavelength = (780.0 750.0 775.0 770.0)      # [nm]
              CoreWidth = 4.0                                   # [micron]
              ModeType = (HEmn EHmn TE0n TM0n)                  # choose mode type
              mModeIndex = (1 1 0 0)                            # m in HEmn, EHmn
              nModeIndex = (1 2 2 1)                            # n in HEmn, EHmn, TE0n, TM0n
              Coordinates = Cylindrical
              Absorption(ODB)
              DiffractionLoss = (5.0 8.0 6.0 7.0)               # [1/cm]
              ModeNumber = 4                                    # solve for 4 modes
         )
      )
      VCSEL()
   )
}
```

Cartesian modes

```
Physics {...
   Laser (...
      Optics (
         EffectiveIndex (
              TargetWavelength = (780.0 750.0 775.0 770.0)      # [nm]
              CoreWidth = 4.0                                   # [micron]
              ModeType = (TE0n TE0n TE0n TE0n)                  # only TE0n allowed for Cartesian
              nModeIndex = (1 2 3 4)                            # n in TE0n
              Coordinates = Cartesian
              Absorption(ODB)
              DiffractionLoss = (5.0 8.0 6.0 7.0)               # [1/cm]
              ModeNumber = 4                                    # solve for 4 modes
         )
      )
      VCSEL()
   )
}
```

This example shows the activation of the EIM for multiple cylindrical and Cartesian modes. The more notable aspects of the syntax are:

- The `CoreWidth` is the radius of the circular aperture in `Cylindrical` coordinates or the half-length of the square aperture in `Cartesian` coordinates.
- The keywords `HEmn`, `EHmn`, `TE0n`, and `TM0n` refer to the common optical fiber modes in the cylindrical modes. For the Cartesian modes, only `TE0n` modes are handled.
- The keywords `ModeType`, `mModeIndex`, and `nModeIndex` are used to specify the required modes.

- The keyword `VCSEL` must be included in the `Laser` statement to specify that this is a VCSEL simulation.
- The number of modes to solve is four in this example, and the parameters corresponding to each mode are specified as shown.

Table 15.149 lists the full range of keywords for the EIM solver.

Table 15.149 Arguments for the EffectiveIndex statement

Feature keyword	Parameter/Description	Default value
`Coordinates=<parameter>`	`Cartesian`	No default
	`Cylindrical`	

`ModeType=<parameter>`	`HEmn`
	`EHmn`
	`TE0n`
	`TM0n`
		No default

`mModeIndex=<int>`	Index `m` of the parameter specified with `ModeType` (`m` = cylindrical harmonic order if `Coordinates=Cylindrical`)	0
`nModeIndex=<int>`	Index `n` of the parameter specified with `ModeType`	0
`TargetWavelength=<float>`	Target value for lasing wavelength [nm]	–1.0
`CoreWidth=<float>`	Specifies the square aperture half-width d (for `Coordinates=Cartesian`) or the circular aperture radius R (for `Coordinates=Cylindrical`) respectively [μm]	–1.0
`DiffractionLoss=<float>`	Specifies the diffraction loss [1/cm]	0
`Absorption(ODB)`	Allows users to specify the material loss in the optical database section of the material parameter file	No default
`ModeNumber=<int>`	Number of modes to be calculated	1

A sample output from the EIM solver is shown in Figure 15.91. The peak of the z-resonant mode aligns with the active quantum well region. Two vectorial transverse modes, HE11 and HE21, computed from the optical fiber problem are also shown.

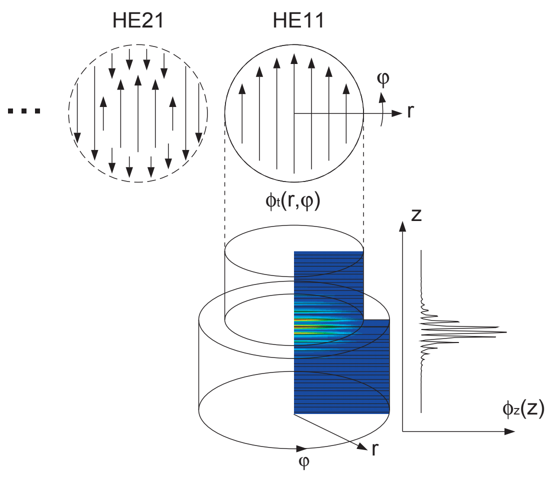


Figure 15.91 Transverse and longitudinal resonant modes of cylindrically symmetric VCSEL computed by EIM

27.8 LED raytracing

Raytracing is used to compute the intensity of light inside a light-emitting diode (LED), as well as the rays that escape from the LED cavity to give the signature radiation pattern for the LED output. The basic theory of raytracing is presented in Section 13.3 on page 15.249.

Due to the random nature of spontaneous emission, the self-consistent solution between the raytracing optics and electronic solver is not possible in LED simulations. Both 2D and 3D LED simulations are possible with DESSIS. The LED simulation uses raytracing mainly to compute the extraction efficiency of the LED, that is, the ratio of light power that escapes from the LED cavity and the total spontaneous emission power.

27.8.1 Isotropic starting rays from spontaneous emission sources

The source of radiation from an LED is mainly from spontaneous emissions in the active region (this is further discussed in Section 28.3.5 on page 15.445). The spontaneous emission in the active region of the LED is assumed to be an isotropic source of radiation and can be conveniently represented by uniform rays emitting from each active vertex, as shown in Figure 15.92.

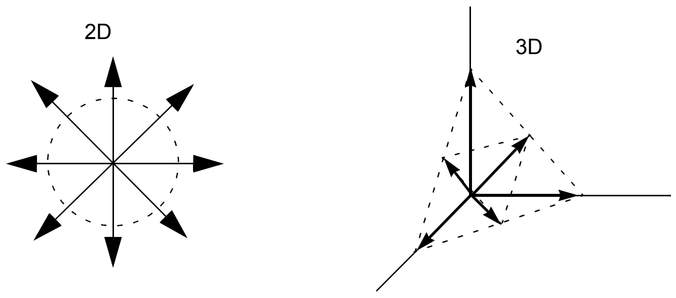


Figure 15.92 Uniform rays radiating isotropically from an active vertex source in 2D (*left*) and 3D (*right*) space: only one-eighth of spherical space is shown for the 3D case

Isotropy requires that the surface area associated with each ray must be the same. The isotropy of the rays in 2D space is apparent. In 3D space, achieving isotropy is not as simple as dividing the angles uniformly. The elemental surface area of a sphere is $r^2 \sin\theta(d\theta)(d\phi)$, so uniformly angular-distributed rays are weighted by $\sin\theta$ and, therefore, do not signify isotropy.

To overcome this problem in 3D, a geodesic dome is used. Rays are directed at the vertices of the geodesic dome such that the surface area associated with each ray is the same. The algorithm starts by constructing an octahedron and, then, recursively splits each triangular face of the octahedron into four smaller triangles. The first stage of this splitting process is shown in Figure 15.92 (*right*), where rays are directed at the vertices of each triangle. The minimum number of rays is six, that is, one directed along each positive and negative direction of the axes. If the first stage of recursive splitting is applied, a few more rays are constructed as shown in Figure 15.92, and the number of starting rays become 18. The second stage of recursive splitting gives 68 rays and so on. Therefore, the user is constrained to selecting a fixed set of starting rays in the 3D case.

27.8.2 Anisotropic starting rays from spontaneous emission sources

In some LED designs, the geometry governs the polarization of the optical field in the device. The spontaneous gain is dependent on the direction of this polarization. Consequently, this will lead to an anisotropic spontaneous-emission pattern at the source.

The anisotropic emission pattern will be described by the following parametric equations:

$$E_x = d1 \cdot \sin(\phi) + d4 \cdot \cos(\phi) \tag{15.518}$$

$$E_y = d2 \cdot \sin(\phi) + d5 \cdot \cos(\phi) \tag{15.519}$$

$$E_z = d3 \cdot \sin(\theta) + d6 \cdot \cos(\theta) \tag{15.520}$$

where the intensity is given by:

$$I = E_x^{\ 2} + E_y^{\ 2} + E_z^{\ 2} \tag{15.521}$$

The bases of sine and cosine are chosen based on the fact that the optical matrix element has such a functional form when polarization is considered (see Section 28.8 on page 15.453). By changing the values of `d1` to `d6`, different emission shapes can be orientated in different directions. Raytracing does not give polarization information, and this feature allows users to modify the anisotropy of the spontaneous emission.

The syntax required to activate the anisotropic spontaneous emission feature is:

```
Physics {...
   LED (...
      Optics(...
         RayTrace(...
            EmissionType(
               #Isotropic              # default
               Anisotropic(
                  Sine(d1 d2 d3)
                  Cosine(d4 d5 d6)
               )
            )
         )
      )
   )
}
```

27.8.3 Randomization of starting rays

Spontaneous emission is a random process. In order to take into account the random nature of this process and still ensure that the emission of the starting rays from each active vertex source is isotropic, a randomized shift of the entire isotropic ray emission is introduced. This is best illustrated in Figure 15.93 on page 15.421 where only four starting rays are used for clarity. For each active vertex, a random angle is generated to determine the random shift of the distribution of the isotropic starting rays. The same concept is also used for the 3D case, and this gives a simple randomization strategy for using raytracing to model the spontaneous emissions.

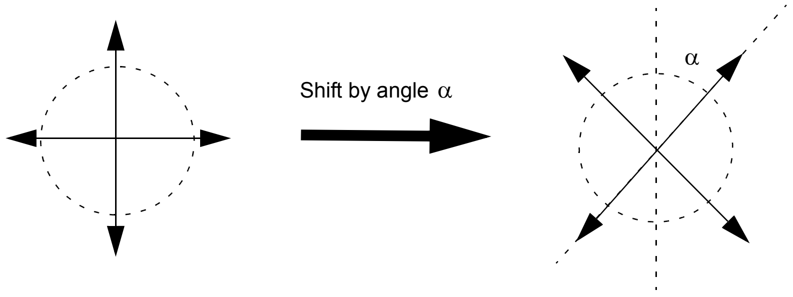


Figure 15.93 Shifting the distribution of entire isotropic starting rays by an angle α

NOTE Randomization of the starting rays is activated by the keyword `RaysRandomOffset` in the `RayTrace` statement. The default is a fixed angular shift, which is determined by the active vertex number.

27.8.4 Syntax for LED raytracing

The raytracing option is activated by the keyword `RayTrace` in the `Physics-LED-Optics` section of the command file. The other sections (`Electrode`, `File`, `Plot`, `Physics`, `Math`, and `Solve`) of the command file for an LED simulation are similar to that of a laser simulation (see Section 25.2.1 on page 15.373 and Section 25.2.2 on page 15.378). Both single-grid and dual-grid LED simulations are possible. The only part that is different is in the definition of the `Physics-LED` section of the command file, as highlighted by this example syntax:

```
Physics {
   AreaFactor = 2       # for symmetric devices
   # ----- Activate LED simulation -----
   LED (
      Optics (
         # ----- Choose ray tracing to compute extraction efficiency -----
         RayTrace(
                  # ----- Info about LED structure -----
                  Symmetry = Symmetric          # or NonSymmetric
                  Coordinates = Cartesian       # or Cylindrical
                  # ----- Specify absorption and refractive index models -----
                  SemAbsorption ( model = ODB )
                  RefractiveIndex( model = ODB )
                  # ----- Specify Starting rays parameters -----
                  RaysPerVertex = 40            # Number of starting rays per active vertex source
                  RaysRandomOffset              # Randomize starting ray angle
                  # ----- Specify ray trace terminating conditions -----
                  DepthLimit = 10               # finish after ray crosses 10 material boundaries
                  MinIntensity = 1e-7           # finish if ray intensity is less than 1e-7

                  LEDRadiationPara(1000.0,180) # (<radius-microns>, Npoints)
                  # ----- Auxiliary features of LED ray tracing -----
#                 Disable                       # disable ray tracing but still run the LED simulation
#                 Print                         # print out all the rays in a grid file
         )
      )

      # ----- Other parameters of the LED structure -----
      Cavitylength = 200                        # for 2D simulation [microns]

      # ----- Choice of spontaneous gain broadening -----
```

```
         Broadening (Type=Lorentzian Gamma=0.10)
   #     Broadening (Type=Landsberg Gamma=0.10)    # Gamma in [eV]
   #     Broadening (Type=CosHyper Gamma=0.10)

         # ----- Specify QW Physics -----
         QWTransport
         QWExtension = AutoDetect                  # auto read QW widths
         QWScatModel
         QWeScatTime = 1e-13                       # [s]
         QWhScatTime = 2e-14                       # [s]
         eQWMobility = 9200                        # [cm^2/Vs]
         hQWMobility = 400                         # [cm^2/Vs]
         # ----- QW strain effects -----
         Strain
         SplitOff = 0.34                           # [eV]
         # ----- Can scale spon gain independently -----
         SponScaling = 1.0
      )

      # ----- User specified physics of transport -----
      Thermionic                  # thermionic emission over interfaces
      HeteroInterfaces            # allow discontinuous bandgap & quasi-Fermi levels
      Mobility ( DopingDep )
      Recombination ( SRH Auger )
      EffectiveIntrinsicDensity ( NoBandGapNarrowing )
      Fermi

      # ------ Option to turn on temperature simulation ------
   # Thermodynamic
   # Hydrodynamic
   # RecGenHeat
   }
```

The `LED` keyword replaces the `Laser` keyword. The arguments for the `LED` statement are generally the same as that for the `Laser` statement (compare this with the `Physics-Laser` section of Section 25.2.1 on page 15.373).

The other notable differences in the LED syntax are:

- The `RayTrace` keyword can only be activated in an `LED` simulation.
- Two types of symmetry are possible: `Symmetric` and `NonSymmetric`. If `Symmetry=Symmetric` is chosen, the user must specify `AreaFactor=2` to account for the correct scaling (see Section 29.3 on page 15.474). If `Coordinates=Cylindrical` is chosen, a `Symmetric` simulation is assumed. The symmetry axis is the y-axis at x = 0.
- The absorption and refractive indices of the material can be specified by the `SemAbsorption` and `RefractiveIndex` keywords. Three choices of model exist: `parameter`, `ODB` and `"pmi_model_name"` as explained in Table 15.150 on page 15.423.
- The keyword `LEDRadiationPara` is associated with the plotting of the LED radiation pattern (see Section 27.8.5 on page 15.423).
- The `Print` keyword outputs all the rays as a grid file. However, if the number of starting rays is large, the output file will be huge, and the high density of rays inside the device will appear as totally black.

NOTE Raytracing can be disabled in an LED simulation by using the keyword `Disable` in the `RayTrace` statement if the user does not require the computing of the extraction efficiency and radiation pattern.

Table 15.150 lists all arguments for the `RayTrace` option and their default values.

Table 15.150 Arguments for RayTrace statement in LED simulation

Feature keyword	Parameter/Description	Default value
`SemAbsorption (model=<parameter>)`	`parameter` (read directly from parameter file)	`parameter`
	`ODB` (use table of values in parameter file)	
	`"pmi_model_name"` (use PMI to input model)	
`RefractiveIndex(model=<parameter>)`	`parameter`	`parameter`
	`ODB`	
	`"pmi_model_name"`	
`depthlimit=<int>`	Number of material boundaries that a ray crosses before the trace is terminated	`5`
`minIntensity=<float>`	Rays are traced until their intensity is less than this number.	`1e-5`
`RaysPerVertex=<int>`	Number of starting rays from each active source vertex. For 3D, the number of starting rays are constrained by 6, 18, 68, and so on. The number in the sequence is chosen such that `RaysPerVertex` is slightly larger or equal to it.	`10`
`Print`	Activates the output of all traced rays to a grid file.	Not activated
`Symmetry=<parameter>`	`Symmetric` `NonSymmetric`	`NonSymmetric`
`Coordinates=<parameter>`	`Cartesian` `Cylindrical`	`Cartesian`
`RaysRandomOffset`	Activates the randomization of the angular shift of the starting rays.	Not activated
`Disable`	Disables raytracing but still runs LED simulation.	Not disabled
`LEDRadiationPara(<float>,<int>)`	The `<float>` specifies the observation radius and `<int>` specifies the discretization of the observation circle or sphere.	No default
`EmissionType(<parameter>)`	`Isotropic`	`Isotropic`

	`Anisotropic( Sine(<float> <float> <float>)`
	`Cosine( <float> <float> <float>) )`	

`LEDSpectrum(<float> <float> <int>)`	The `<float>`s give the starting and ending energy range in [eV] respectively. The `<int>` gives the number of discretizations between in that energy range.	no default

27.8.5 LED radiation pattern

Raytracing does not contain phase information, so it is not possible to compute the far-field pattern for an LED structure. Instead, the outgoing rays from the LED raytracing are used to produce the radiation pattern. In 2D space, this is equivalent to moving a detector in a circle around the LED as shown in Figure 15.94 on page 15.424. In 3D space, the detector is moved around on a sphere. The circle and sphere have centers that correspond to the center of the device. DESSIS automatically determines the center of the device to be the midpoint of the device on each axis.

To examine the optical intensity inside the LED, the user can use either `OpticalIntensityMode0` in the `Plot` statement or the `SaveOptField` option in the `File` statement (see Section 29.2 on page 15.472).

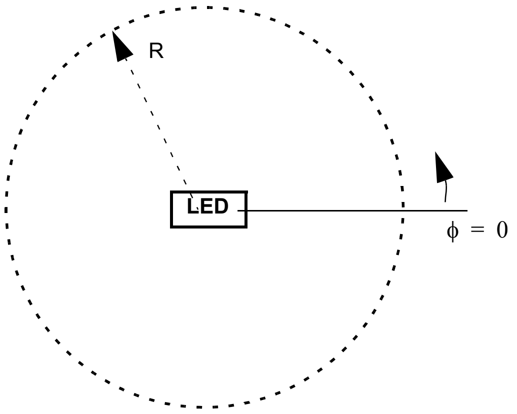


Figure 15.94 Measuring the radiation pattern in a circular path around LED at observation radius, R

The syntax required to activate and plot the LED radiation pattern is located in the `File`, `Physics-LED-Optics-RayTrace`, and `Solve-quasistationary` sections of the command file:

```
File {...
   # ----- Activate LED radiation pattern and save -----
   LEDRadiation = "rad"

}
...
Physics {...
   LED (...
      Optics (...
         RayTrace(...
                  LEDRadiationPara(1000.0,180) # (<radius-microns>, Npoints)
         )
      )
   )
}
...
Solve {...

   # ----- Specify quasistationary -----
   quasistationary (...

        PlotLEDRadiation { range=(0,1) intervals=3 }

        Goal {name="p_Contact" voltage=1.8})
        {...}
}
```

The LED radiation plot syntax works in the same way as the `GainPlot` (see Section 29.4 on page 15.475) and the `OptFarField` plot (see Section 27.9 on page 15.427) in the `Quasistationary` statement.

An explanation of this example is:

- The base file name, `"rad"`, of the LED radiation pattern files is specified by `LEDRadiation` in the `File` section. The keyword `LEDRadiation` also activates the LED radiation plot.
- The parameters for the LED radiation plot are specified by the keyword `LEDRadiationPara` in the `Physics-Optics-RayTrace` section. The user must specify the observation radius (in microns) and the discretization of the observation circle (2D) or sphere (3D).
- The LED radiation pattern can only be computed and plotted within the `Quasistationary` statement. The keyword `PlotLEDRadiation` controls the number of LED radiation plots to produce.
- The argument `range=(0,1)` in the `PlotLEDRadiation` keyword is mapped to the initial and final bias conditions. In this example, the initial and final (goal) `p_Contact` voltages are 0 V and 1.8 V, respectively. The number of `intervals=3`, which gives a total of four (= 3+1) LED radiation plots at 0 V, 0.6 V, 1.2 V, and 1.8 V. In general, specifying `intervals=n` will produce (n+1) plots.
- If the LED structure is symmetric, the LED radiation is only computed on a semicircle.

The following briefly describes the files that are produced in the LED radiation plot, for the 2D and 3D cases.

Two-dimensional LED radiation pattern and output files

Activating the LED radiation plot for a 2D LED simulation produces four different files (using the base name "`rad`"):

`rad.grd`	Discretized grid of a circle with polar coordinates (r, ϕ) in DF–ISE format.
`rad_000000_LEDRad.dat`	Data file containing the normalized radiation pattern in DF–ISE format to be used in conjunction with `rad.grd`.
`rad_000000_LEDRad.plt`	The normalized radiation pattern versus observation angle, which can be viewed in INSPECT.
`rad_000000_LEDRad_Polar.grd`	The normalized radiation pattern projected onto a grid file and can be viewed in Tecplot-ISE. Run Tecplot-ISE and load the file `rad_000000_LEDRad_Polar.grd`. Select **Data** > **Alter** > **Transform Coordinates**, and select the transformation **Polar to Rectangular**. Assign `Source Theta = Y` and `Source R = X`. Click **Compute** and click **Close**. The polar plot of the LED radiation pattern is shown.

A sample output of the radiation plot of a 2D nonsymmetric LED structure is shown in Figure 15.95. The lower-left image corresponds to the file `rad_000000_LEDRad.plt` plotted by INSPECT, and the right image is the product of the file `rad_000000_LEDRad_Polar.grd` plotted by Tecplot-ISE.

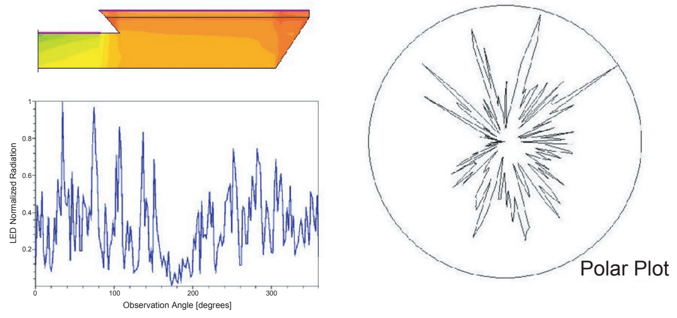


Figure 15.95 LED internal optical intensity (*upper left*), normalized radiation intensity versus observation angle (*lower left*), and polar radiation plot (*right*) computed by DESSIS in 2D LED simulation

Three-dimensional LED radiation pattern and output files

There are only two output files for the radiation pattern in the case of a 3D LED simulation:

`rad.grd`	Discretized grid of a sphere with spherical coordinates (r, ϕ, θ) in DF–ISE format.
`rad_000000_LEDRad.dat`	Data file containing the normalized radiation pattern in DF–ISE format to be used in conjunction with `rad.grd`. As Tecplot-ISE treats all grid input and displays the output as Cartesian coordinates, it is necessary to transform the data so that the spherical coordinates data can be viewed. This is performed with the same steps as in the 2D case: Run Tecplot-ISE and load the files `rad.grd` and `rad_000000_LEDRad.dat`. Select **Data** > **Alter** > **Transform Coordinates**. Select the transformation **Spherical to Rectangular**. Assign `Source Theta=Z`, `Source R=X`, and `Source Psi=Y`. Click **Compute** and click **Close**. The spherical plot of the 3D LED radiation pattern is shown. A sample of the radiation pattern of a 3D LED simulation is in Figure 15.96.

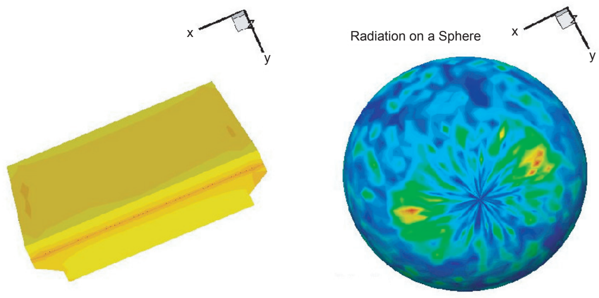


Figure 15.96 LED internal optical intensity (*left*) and normalized radiation intensity projected on a sphere (*right*) of 3D LED simulation

27.8.5.1 Spectrum-dependent LED radiation pattern

Unlike a laser beam with single-frequency emissions, rays emitting from an LED carry a spectrum of frequencies (or energies). DESSIS monitors the spectrum of each ray as it undergoes the process of raytracing in and out of the device. The resultant spectrum of the LED radiation pattern can then be plotted.

To activate this feature, users must include the keyword `LEDSpectrum` in the command file:

```
Physics {...
   LED (...
      Optics (...
         RayTrace(...
            LEDSpectrum(<startenergy> <endenergy> <numpoints>)
         )
      )
   )
}
```

This feature must be used in conjunction with the `LEDRadiation` feature so that the file names of the LED radiation plots and the observation angles can be specified. Other notable aspects of the syntax are:

- `<startenergy>` and `<endenergy>` give the energy range of the spectrum to be monitored. These parameters are floating-point entries with units of eV.
- `<numpoints>` is an integer determining the number of discretized points in the specified energy range.

27.9 Far field

The far field is important to determine the beam divergence of the laser diode emission. From antenna theory, the far field is defined for observation distance:

$$r \geq \frac{2D^2}{\lambda} \tag{15.522}$$

where D is the largest dimension of the near-field shape and λ is the free space wavelength. For example, given a near-field shape of dimension 2×2 µm and a wavelength of 1 µm, the far field occurs at the observation distance $r \geq 8$ µm.

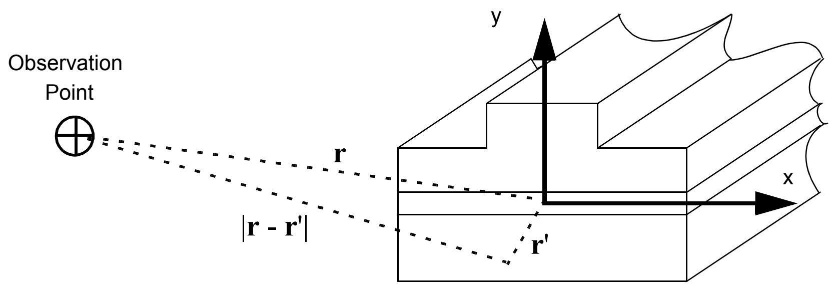


Figure 15.97 Origin is conveniently placed at center of laser end facet so that z'=0; distance between the observation point and a source point on laser end facet is $|\boldsymbol{r} - \boldsymbol{r}'|$

Figure 15.97 is referred to for the coordinate used in the following derivation of the vectorial far field. The optical electric field at the observation distance r [132] is:

$$\boldsymbol{E}(\boldsymbol{r}) = -\int [\nabla \times \vec{\boldsymbol{G}}(\boldsymbol{r}, \boldsymbol{r}') \bullet \boldsymbol{M}(\boldsymbol{r}')] d\boldsymbol{r}' \tag{15.523}$$

where $\vec{\boldsymbol{G}}$ is the dyadic Green operator and $\boldsymbol{M}$ is an equivalent magnetic source representing the near field, $\boldsymbol{E}_{near}(\boldsymbol{r}')$, of the laser mode:

$$\boldsymbol{M}(\boldsymbol{r}') = -2\hat{z} \times \boldsymbol{E}_{near}(\boldsymbol{r}') \tag{15.524}$$

In the far field, it is assumed that the radiation becomes plane waves, and the curl of the dyadic Green operator can be simplified to:

$$\nabla \times \vec{\boldsymbol{G}}(\boldsymbol{r}, \boldsymbol{r}') = ik\hat{\boldsymbol{r}} \times \vec{\boldsymbol{G}}(\boldsymbol{r}, \boldsymbol{r}') \tag{15.525}$$

where the unit vector:

$$\hat{\boldsymbol{r}} = \hat{\boldsymbol{x}}\sin\theta\cos\phi + \hat{\boldsymbol{y}}\sin\theta\sin\phi + \hat{z}\cos\theta \tag{15.526}$$

In addition, the following approximations are assumed for far-field calculations:

$\lvert\boldsymbol{r} - \boldsymbol{r}'\rvert \approx r$	for amplitude
$\lvert\boldsymbol{r} - \boldsymbol{r}'\rvert \approx r - \hat{\boldsymbol{r}} \bullet \boldsymbol{r}'$	for phase

With these assumptions, formulas for the vectorial and scalar far-field calculations can be derived.

In the vectorial case, simplifying the dyadic Green operator with the scalar Green function and setting $z' = 0$, the vectorial far field can be derived as:

$$\boldsymbol{E}(r, \theta, \phi) = -\frac{ike^{ikr}}{4\pi r}\left\{\hat{\boldsymbol{x}}2\cos\theta\iint E_{near(x)}(x', y')e^{-ik\hat{r}\bullet\bar{r}'}dx'dy'\right. \tag{15.527}$$

$$+\hat{\boldsymbol{y}}2\cos\theta\iint E_{near(y)}(x', y')e^{-ik\hat{r}\bullet\bar{r}'}dx'dy'$$

$$\left.-\hat{z}2\sin\theta\iint[E_{near(x)}(x', y')\cos\phi + E_{near(y)}(x', y')\sin\phi]e^{-ik\hat{r}\bullet\bar{r}'}dx'dy'\right\}$$

where $E_{near(x)}$ and $E_{near(y)}$ are the x and y components of the near field, respectively.

Using the transformations:

$$\sin\Theta_x = \sin\theta\cdot\cos\phi \tag{15.528}$$

$$\sin\Theta_y = \sin\theta\cdot\sin\phi \tag{15.529}$$

users can derive from (Eq. 15.527) the commonly used expression for the constant radius, relative far-field intensity:

$$I_{far}(\Theta_x, \Theta_y) = (1-(\sin^2\Theta_x + \sin^2\Theta_y))\left|\iint\Phi(x, y)e^{ik_0(x\sin\Theta_x + y\sin\Theta_y)}dxdy\right|^2 \tag{15.530}$$

In (Eq. 15.530), it is clear that the scalar near field, Φ, can represent either $E_{near(x)}$ (TE mode) or $E_{near(y)}$ (TM mode) in (Eq. 15.527).

NOTE If the square root of the optical intensity is used to compute the far field, the phase information will be lost and the far field will be incorrect. Therefore, it is important to ensure the scalar or vectorial near fields are used in the far-field calculations.

27.9.1 Far-field observation angle

The mapped far-field observation angles, (Θ_x, Θ_y), refer to the angles measured from the z-axis along the x-axis and y-axis, respectively. The laser end facet (location of the near field) is assumed to be at the origin, facing the direction of positive z (see Figure 15.98 on page 15.429).

The far-field observation distance is fixed at a constant radius from the origin where the device is, that is, the observation space is a hemisphere. As a result, constraints must be imposed on (Θ_x, Θ_y) because the aim is to map a rectangular (Θ_x, Θ_y) $[-\pi/2, \pi/2]$ space onto a hemisphere. The best way to visualize this constraint is to think of placing a square piece of fishnet over a hemisphere. The corner regions of the net will exceed the boundary of the hemisphere and, hence, do not contribute to the description of the hemisphere. These are the invalid zones of the observation space.

NOTE In the far-field calculation, the origin is aligned with the peak intensity of the fundamental mode of the edge-emitting laser, and the observation angles are defined with respect to this origin.

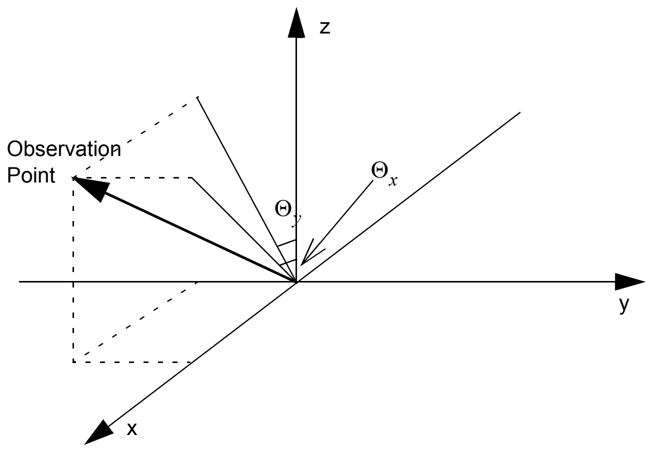


Figure 15.98 Defining the observation angles for far field

27.9.2 Syntax of far field

The optical far field is computed and plotted if the keyword `OptFarField` is specified in the `File` section of the command file:

```
File {...
   # ----- Activate farfield computation and save -----
   OptFarField = "far"

}
...
Solve {...

   # ----- Farfield plot parameters must be specified inside quasistationary -----
   quasistationary (...

        PlotFarField{range=(0,1) intervals=3}
        PlotFarFieldPara{range=(40,60) intervals=40 Scalar1D Scalar2D Vector2D}

        Goal {name="p_Contact" voltage=1.8})
        {...}
}
```

See Section 25.2.1 on page 15.373 for a clearer picture of the placement of the far-field syntax inside the command file. The far field syntax works in the same way as the `GainPlot` (see Section 29.4 on page 15.475) and `Plot` options in the `Quasistationary` statement. The more notable features of the syntax are:

- The base file name `"far"` of the far-field files is specified by `OptFarField` in the `File` statement. The keyword `OptFarField` also activates the far-field plot.
- The far field can only be computed and plotted within the `Quasistationary` statement. The keywords `PlotFarField` and `PlotFarFieldPara` control the number and type of far field plots to produce.
- The argument `range=(0,1)` in the `PlotFarField` keyword is mapped to the initial and final bias conditions. In this example, the initial and final (goal) `p_Contact` voltages are 0 V and 1.8 V, respectively. The number of `intervals=3`, which gives a total of four (= 3+1) far-field plots at 0 V, 0.6 V, 1.2 V, and 1.8 V. In general, specifying `intervals=n` will produce (n+1) plots.

- The argument `range=(40,60)` in the `PlotFarFieldPara` keyword is the range for the observation angles. In this example, $\Theta_x = [-20°, 20°]$ and $\Theta_y = [-30°, 30°]$ were chosen. The number of discretized points in each range of the observation angles is given by `intervals=40`.
- Users can select any one or a combination of the three types of far-field plot: `Scalar1D`, `Scalar2D`, and `Vector2D`. These keywords correspond to the scalar one-dimensional far field, scalar two-dimensional far field, and vectorial two-dimensional far field, respectively. The different types of far-field plot will generate different types of output files.

Table 15.151 and Table 15.152 summarize the activating syntaxes for the far field and the naming convention for the output files for different far-field plots.

Table 15.151 Specifying far-field base name in File statement and naming convention for far-field output files

Feature keyword	Description
`optfarfield ="<string>"`	`<string>` is used as the file prefix for various output files: `<string>_ff_<number>_des.plt` for 1D plots. `<string>_ff_des.grd` for 2D observation angle grid. `<string>_ff_des_<number>_Scalar.dat` for 2D scalar far-field data file. `<string>_ff_des_<number>_Vector.dat` for 2D vector far-field data file.

Table 15.152 Arguments for plotting far field in Quasistationary statement

Feature keyword	Description
`plotfarfield{range=(0, 1) interval=<int>}`	Notes the number of times far-field plots are made during the quasistationary solve process.
`plotfarfieldpara{range=(h_angle, v_angle) interval=<int> Scalar1D | Scalar2D | Vector2D}`	`(h_angle, v_angle)` are the horizontal and vertical angle ranges. `<int>` specifies the number of points per axis. The type of far field plot is chosen by `Scalar1D`, `Scalar2D`, and `Vector2D`. If no type is chosen, the default `Scalar1D` is used.

27.9.3 Far-field output files

Activating different types of far-field plot produces different output files. Referring to the example syntax listed in Section 27.9.2 on page 15.429, where the far-field base name has been specified by `OptFarField="far"`, output examples where different far-field types are selected are shown.

Scalar1D far field

The scalar 1D far field is activated by the keyword `Scalar1D`. Two lines per mode are produced for the 1D far-field plot. One is the far-field intensity measured along Θ_x with $\Theta_y = 0$; the other, along Θ_y with $\Theta_x = 0$. These far field results are output in the files `far_ff_000000_des.plt`, `far_ff_000001_des.plt`, and so on at the requested bias `intervals` specified in the argument of `PlotFarField`. These files can be viewed in INSPECT and an example of the plot is shown in Figure 15.99 on page 15.431.

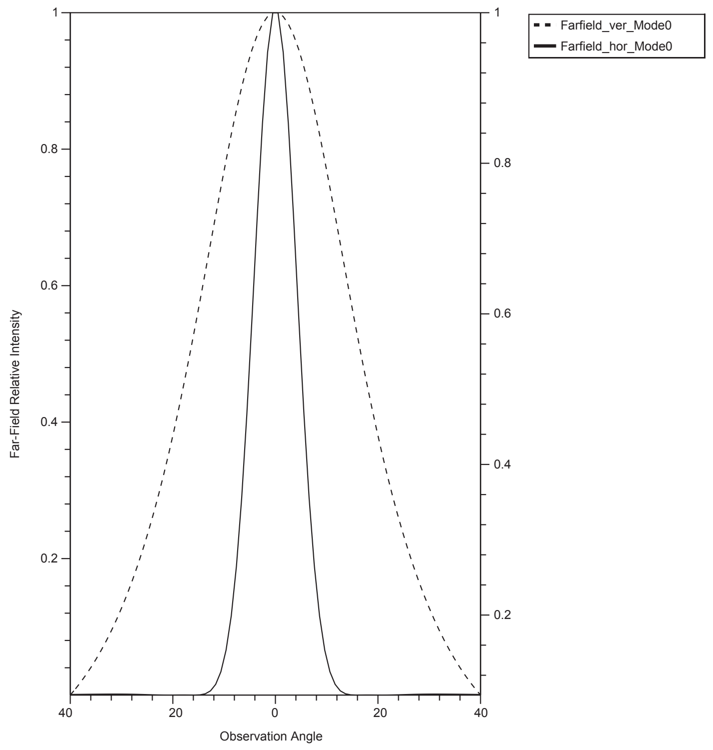


Figure 15.99 Scalar 1D far-field patterns showing vertical (*dashed line*) and horizontal (*solid line*) variations

Scalar2D and Vector2D far fields

The 2D scalar and vectorial far fields are activated by the keywords `Scalar2D` and `Vector2D`, respectively. An observation angle grid, `far_ff_des.grd`, is created if either keyword is detected. The vectorial far field can only be activated if the vectorial optical solver has been chosen or a vectorial mode is input from a file.

The data file for the scalar 2D far field contains the normalized far-field pattern of each mode, and the file names are `far_ff_des_000000_Scalar.dat`, and so on. Each file contains as many far-field patterns as the number of modes in the simulation. If a vectorial optical solver has been chosen, DESSIS automatically selects the strongest field component (either E_x or E_y) for its scalar far-field computation.

The data file for the vectorial 2D far field contains the normalized far-field pattern for the x, y, and z components and the total far field for each mode. These data files are named as `far_ff_des_000000_Vector.dat`, and so on. The 2D scalar and vectorial far fields can be viewed in Tecplot-ISE like any other `.grd` and `.dat` files. Examples of these far-field patterns are shown in Figure 15.100 and Figure 15.101 on page 15.432.

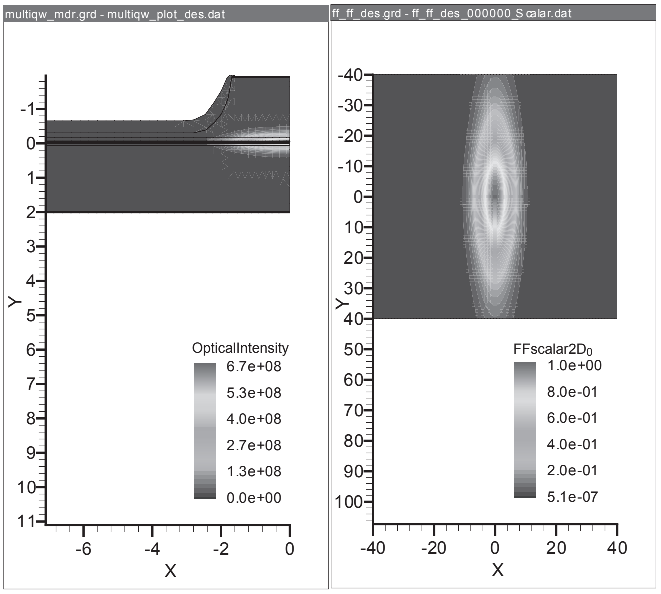


Figure 15.100 Near field (*left*) and scalar 2D far field (*right*) of edge-emitting laser

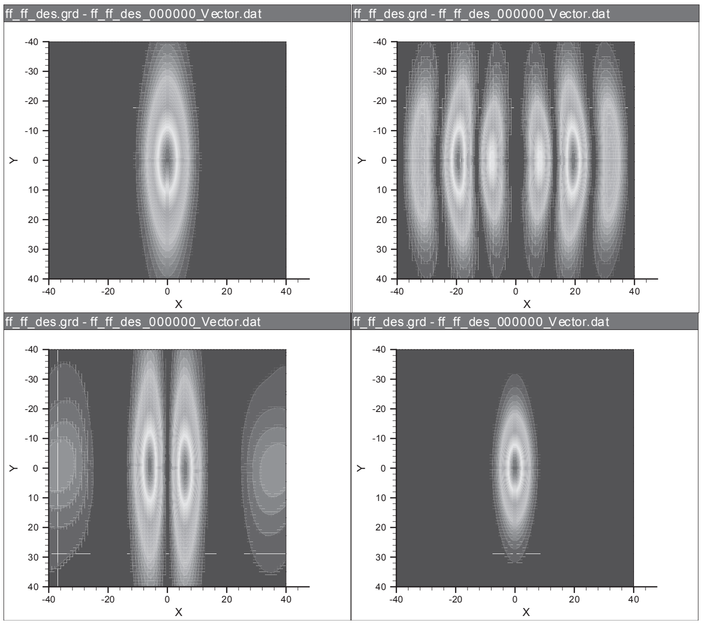


Figure 15.101 The x (*upper left*), y (*upper right*), z (*lower left*) components, and total absolute value (*lower right*) of vectorial 2D far fields of edge-emitting laser; x component is a few orders of magnitude stronger than y and z components, so the absolute of the vectorial far field resembles that of x component

27.9.4 Far field from loaded optical field file

Far fields can be computed from the optical mode loaded into DESSIS (see Section 29.2 on page 15.472). To compute the far field correctly, the scalar or vectorial optical near field must be loaded. The optical intensity does not contain phase information and will give an incorrect far-field pattern.

27.10 VCSEL near field and far field

While the near field of an edge emitter is apparent, the location of the near field of a VCSEL is not automatically detected in DESSIS. Different VCSELs have different geometric designs and the entire top surface may not be the emitting surface. Therefore, in the case of VCSELs, the user must specify the location of the near field that will be used to compute the far field.

The activating syntaxes for the VCSEL near field are in the `File`, `Physics-Laser`, and `Solve-quasistationary` sections of the command file:

```
File {...
      VCSELNearField = "vcselnf"
}
...
Physics {...
   Laser (...
      Optics (...
         FEVectorial(...)
      )
      VCSEL (...
         NearField(10.0 0.0 100)        # (<radius> <z> Npoints) [microns]
      )
   )
}
...
Solve {...
   # ----- VCSEL near field parameters must be specified inside quasistationary -----
   quasistationary (...
      VCSELNearField { range=(0,1) intervals=3 }
      Goal({name="p_Contact" voltage=1.8})
      {...}
}
```

An analysis of the syntax is:

- In the `File` section, the VCSEL near field is activated by the keyword `VCSELNearField`, and the value assigned to it, `"vcselnf"` in this case, is the base name for the output files of the near field.
- In the `Physics-Laser-VCSEL` section, the location and discretization mesh of the near field is defined by the keyword `NearField(<radius> <z> Npoints)`. In VCSELs where cylindrical geometry has been assumed, `<radius>` and `<z>` (in microns) define the location of the near field, that is, the near field is taken from (0,`<z>`) to (`<radius>`,`<z>`). `Npoints` defines the number of points on each side of the square mesh where the near field will be plotted.
- In the `Solve-quasistationary` section, the argument `range=(0,1)` in the `VCSELNearField` keyword is mapped to the initial and final bias conditions. In this example, the initial and final (goal) `p_Contact` voltages are 0 V and 1.8 V, respectively. The number of `intervals=3`, which gives a total of four (= 3+1) near field plots at 0 V, 0.6 V, 1.2 V, and 1.8 V. In general, specifying `intervals=n` will produce (n+1) plots.
- The square mesh grid file for the near field will be named `vcselnf.grd`. At each requested bias output, four files will be created for each mode: `vcselnf_00000n_mode0_int_even.dat`, `vcselnf_00000n_mode0_int_odd.dat`, `vcselnf_00000n_mode0_real.dat`, and `vcselnf_00000n_mode0_imag.dat`. The near fields can be viewed in Tecplot-ISE by loading, for example, `tecplot_ise vcselnf.grd vcselnf_000000_mode0_int_even.dat`.

- Cylindrical symmetry has been assumed so that the optical fields decompose into cylindrical harmonics, $e^{im\phi}$, where m is the cylindrical harmonic order defined by the keyword `AzimuthalExpansion` in a VCSEL simulation. The near-field intensity is divided into the even part (multiplied by $\cos(m\phi)$) and the odd part (multiplied by $\sin(m\phi)$). For the real and imaginary vectorial fields, the cylindrical harmonic $e^{im\phi}$ is multiplied by the complex optical field, and the real and imaginary parts of the resultant field are saved.

When the VCSEL near field is computed, the activation of the far-field calculations is similar to that of other laser simulations.

NOTE In the case of VCSELs, the far field is not computed unless the near field is specified.

27.11 Automatic optical mode searching

The cavity and waveguide mode solvers in DESSIS require the user to choose an initial guess so that a sparse matrix solver can be used to find the correct optical mode. It is not always easy to estimate a good initial guess. To circumvent this problem, a model has been developed to help identify the probability of physical modes through a mode density calculation. The concept is simple: A Hertzian dipole is placed in the cavity or waveguide, and the deterministic equations:

$$\nabla\times(\nabla\times\bar{E}) - \frac{\omega^2}{c^2}\varepsilon_r\bar{E} = -i\omega\mu\cdot\dot{j}_{src}(r,\omega) \quad (15.531)$$

are solved for the cylindrically symmetric cavity resonance problem and:

$$\nabla\times(\nabla\times\bar{E}) - \frac{\omega^2}{c^2}\varepsilon_r(x,y)\cdot\bar{E}(x,y) + \gamma^2\cdot\bar{E}(x,y) = -i\omega\mu\cdot\dot{j}_{src}(r,\omega) \quad (15.532)$$

are solved for the waveguide problem. The source dipole $\dot{j}_{src}(r,\omega)$ is placed strategically in the cavity (see Figure 15.102) or waveguide (see Figure 15.103 on page 15.435).

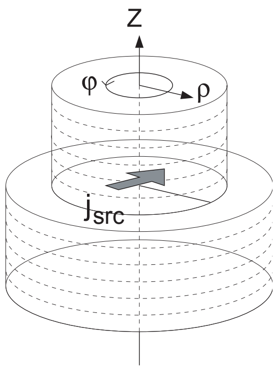


Figure 15.102 Dipole source in a cylindrically symmetric cavity problem can be radially or azimuthally defined

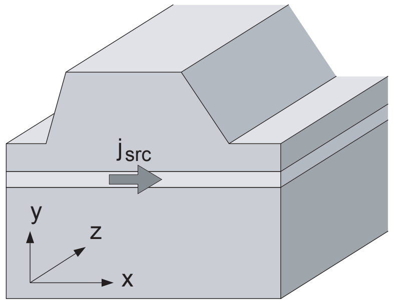


Figure 15.103 Dipole source in a waveguide problem can be orientated in x or y direction to excite TE and TM modes

The source dipole will excite a plethora of modes containing guided, resonating, and leaky modes. By sweeping through the wavelengths for the cavity problem or the effective indices for the waveguide problem, the change of energy in the cavity or waveguide at each frequency or effective index can be computed. This energy change is most sensitive near a resonating or guided mode and, therefore, provides the key to identifying the physical resonating or guided modes. The total energy contained in the system is given by:

$$\int\int_{surf} (\bar{S} \cdot d\bar{A}) = -\frac{1}{2}\iiint_V \{\sigma\bar{E} \cdot \overline{E^*} + i\omega(\mu\bar{H} \cdot \overline{H^*} + \varepsilon\bar{E} \cdot \overline{E^*})\}dV \qquad (15.533)$$

The real part gives the physically dissipated energy of the system and the imaginary part gives the energy stored in the system. In any system containing resonances, it is well known that the energy dissipation at the resonance point reduces sharply to a minimum. Therefore, by tracking the changes in dissipated energy, it is possible to estimate the resonant wavelength or effective index of the required mode.

There are remote possibilities that the dipole source is placed at the null of the required mode. In this case, the mode will not be excited and will not show up in the plot of the change in dissipation energy. Users can place the dipole source at alternative positions to ensure that the required mode is excited.

27.11.1 Syntax for automatic mode searching

There are some differences in the syntax for the automatic mode search for the cavity and waveguide problems.

27.11.1.1 Searching for cavity resonances

Cylindrical symmetry is assumed for the cavity problem. This is applicable to the case of cylindrically symmetric VCSEL structures. The syntax for setting up the deterministic search for cavity resonances is:

```
File {
   ...
   SaveOptSpectrum = "spectrum"
}
...
Physic {
   Laser (
      Optics (
         DeterministicProblem (
            EquationType = Cavity
```

```
                Coordinates = Cylindrical
                AzimuthalExpansion = 1
                SourcePosition (0.25 0.5)              # [um]
                SourcePolarization = rhoDirection
                Sweep (740.0 760.0 80)                 # [nm]
            )
        )
        VCSEL()
    )
    ...
}
....
Solve { Optics }
```

The `SourcePolarization` of the dipole can be chosen to be in different directions. A summary of these directions is given in Table 15.153 on page 15.437. The range of wavelengths to search is given in units of nanometers. An example of the output plot is shown in Figure 15.104. As expected, the resonances occur at locations with sharp energy changes.

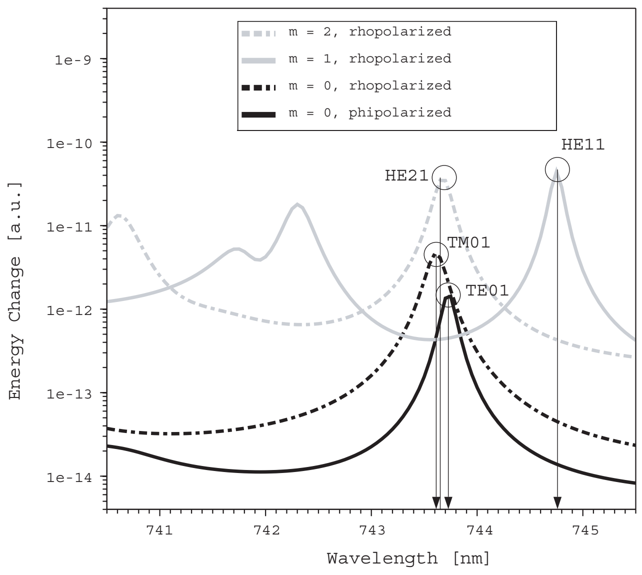


Figure 15.104 Energy changes plotted as a function of wavelength for the cavity problem; resonances of the different modes are clearly identified

27.11.1.2 Searching for waveguide modes

Searching for waveguide modes requires sweeping through the effective indices, and the syntax for this is:

```
File {
     SaveOptSpectrum = "spectrum"
}
...
Physics {
    Laser (
        Optics (
            DeterministicProblem (
                EquationType = Waveguide
                Symmetry = Symmetric
                Boundary = Type1
                Coordinates = Cartesian
                LasingWavelength = 530      # [nm]
                SourcePosition (1.0 1.533)
                SourcePolarization = xDirection
                Sweep (3.391 3.401 100)
             )
          )
       )
    }
    ...
    Solve { Optics }
```

An example of the output is shown in Figure 15.105.

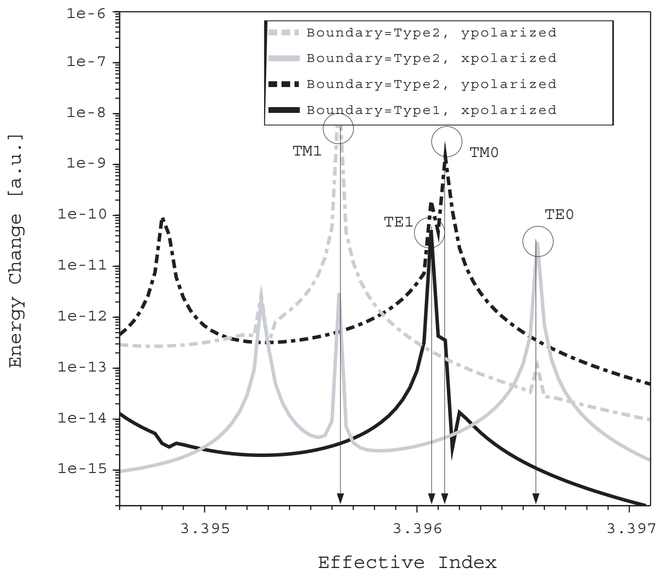


Figure 15.105 Energy changes plotted as a function of the effective index; guided modes of the waveguide are clearly identified

27.11.1.3 Summary of keywords

Table 15.153 summarizes the keywords for the `DeterministicProblem` section.

Table 15.153 Keywords for DeterministicProblem section

Keyword	Parameter/Description
`EquationType = <parameter>`	`Cavity`
	`Waveguide`
`Coordinates = <parameter>`	`Cylindrical` (for cavity only)
	`Cartesian` (for waveguide only)
`AzimuthalExpansion = <int>`	Specifies order of the cylindrical harmonics, $\cos(m\varphi)$ (for cavity only)
`LasingWavelength = <float>`	Lasing wavelength [nm] (for waveguide only)

Table 15.153 Keywords for DeterministicProblem section

Keyword	Parameter/Description
`Symmetry = <parameter>` (for waveguide only)	`Symmetric`
	`Periodic`
	`NonSymmetric`
`Boundary = <parameter>` (for waveguide only)	`Type1`
	`Type2`
`SourcePosition(<float> <float>)`	Coordinates of the source dipole (x-coordinate, y-coordinate)
`SourcePolarization = <parameter>`	`rhoDirection` (for cavity only)
	`phiDirection` (for cavity only)
	`zDirection`
	`xDirection` (for waveguide only)
	`yDirection` (for waveguide only)
`Sweep(<float> <float> <int>)`	`(<start-wavelength> <end-wavelength> <Npoints>)` [nm] (for cavity)
	`(<start-eff-index> <end-eff-index> <Npoints>)` [1] (for waveguide)

CHAPTER 28 Quantum well modeling

28.1 Overview

The drift-diffusion transport phenomena contained in the continuity and hydrodynamic equations are not suitable for modeling the transport across the quantum well (QW) because the feature size of the quantum well is much smaller than the inelastic mean free path of the carrier.

In this section, the focus is on three aspects of modeling the quantum well:

- Carrier capture into the quantum well
- Radiative recombination processes important in a quantum well
- Gain calculations

The QW carrier capture process is treated with a ballistic approach. A few types of recombination processes are important in the quantum well:

- Auger and Shockley–Read–Hall (SRH) recombinations deplete the QW carriers, and they form the dark current.
- Radiative recombination contains the stimulated and spontaneous recombination processes, which are important processes in lasers and LEDs.

These recombinations must be added to the carrier continuity equations to ensure the conservation of particles.

The gain calculation is based on Fermi's golden rule and describes quantitatively the radiative emissions in the form of the stimulated and spontaneous emission coefficients. These coefficients contain the optical matrix element $|M_{ij}|^2$, which describes the probability of the radiative recombination processes. In the quantum well, computing the optical matrix element requires knowledge of the QW subbands and QW wavefunctions.

DESSIS offers three options for computing the gain spectrum:

- A simple finite well model with analytic solutions. In addition, strain effects and polarization dependence of the optical matrix element are handled separately.
- The k.p method, which includes strain and many-body effects with its 4×4, 6×6, or 8×8 Hamiltonian matrix.
- User-specified gain routines in C++ language can be coupled self-consistently with DESSIS using the physical model interface (PMI).

28.2 Carrier capture in quantum wells

A ballistic transport approach is used to handle the carrier capture or escape into or out of a quantum well. It follows the treatment of Grupen and Hess [133]. This approach is illustrated in Figure 15.106.

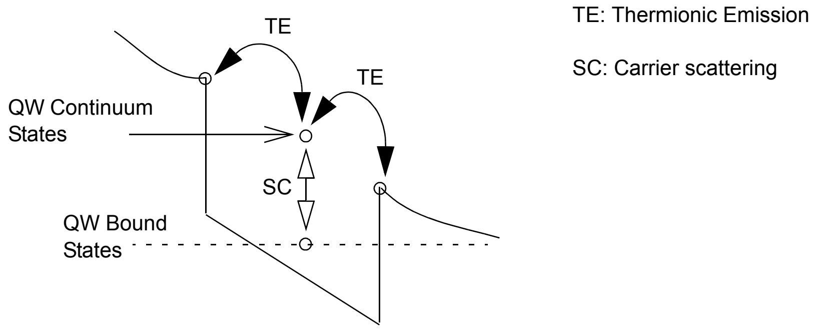


Figure 15.106 Discretization of quantum well to handle the physics of quantum well transport

The transport perpendicular to the QW plane is treated as follows. The QW is treated as a point source of recombination, which contains separate continuum and bound states. In this case, the continuum and bound states are assumed to have different quasi-Fermi levels that lead to separate continuity equations for the continuum and bound states. Transport of carriers from the regions outside the QW to the QW continuum states is by thermionic emission. The transition from the QW continuum to the bound states is computed by a scattering rate that includes carrier–carrier scattering. The bound states are solved from the Schrödinger equation.

Within the quantum well in the direction parallel to the QW plane, drift-diffusion transport is assumed to be valid.

28.2.1 Special meshing requirements for quantum wells

As a result of the transport in Figure 15.106, the spatial discretization of the mesh at the quantum well must be treated specially. The discretization is based on a 'three-point' model in the direction perpendicular to the QW plane. Only one vertex is placed in the quantum well and one vertex is at each quantum well–bulk interface (see Figure 15.107). In the quantum well, only rectangular elements are allowed. This must be ensured when building the mesh.

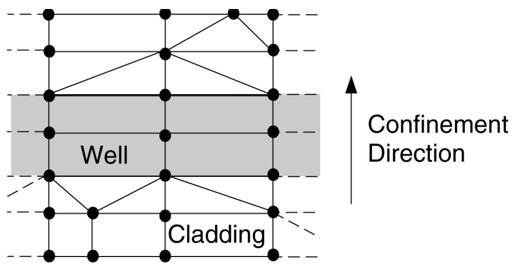


Figure 15.107 Discretization of quantum well

NOTE The discretization constraint for the quantum well can be chosen by creating a refinement region at the quantum well in the mesh generator.

28.2.2 Thermionic emission

Thermionic emission is used at the quantum well interface to handle the large momentum changes caused by the band-edge discontinuity. If thermionic emission is used only, without the QW scattering model, all the carriers are assumed to be totally captured in the quantum well.

The default setting uses thermionic emission at the quantum well interfaces. This is activated by default if the keyword `QWTransport` is used in the `Physics-Laser` section of the command file:

```
Physics {...
   Laser (...
      Optics (...)
      # ----- Specify QW model and physics -----
      QWTransport
      QWExtension = AutoDetect# QW widths auto detection
   )
}
```

The QW region must be identified by the keyword `Active` (see Section 25.2.1 on page 15.373) and must be discretized according to the constraints of the special ‘three-point’ QW model. The keyword `QWTransport` informs DESSIS to use the ‘three-point’ QW model to treat the `Active` quantum well region.

It is possible to use the thermionic emission QW model without the QW scattering model for isothermal quasistationary simulations, in which case, all carriers are assumed to be totally captured in the quantum well. However, excluding the QW scattering model in nonisothermal simulations (that is, when the hydrodynamic or thermodynamic temperature equations are included) will lead to convergence problems.

28.2.3 QW scattering model

A physically intuitive model is used to handle the physics of carrier scattering at the quantum well. The carrier populations are separated into bound and continuum states, and separate continuity equations are applied to both populations. The QW scattering model accounts for the net capture rate, that is, not all of the carriers will be scattered into the bound states of the quantum well.

The electron capture rate from the continuum (subscript 3) to the bound (subscript 2) states is:

$$R = \int_{E_C}^{\infty} dE_3 \int_{E_{well}}^{\infty} dE_2 \cdot g_3(E_3) g_2(E_2) S(E_2, E_3) f_3(E_3)(1 - f_2(E_2)) \tag{15.534}$$

where $g(E)$ is the density of states, $S(E_2, E_3)$ is the scattering probability, and $f(E)$ is the Fermi–Dirac distribution. The reverse process gives the electron emission rate from the bound to continuum states:

$$M = \int_{E_C}^{\infty} dE_3 \int_{E_{well}}^{\infty} dE_2 \cdot g_3(E_3) g_2(E_2) S(E_3, E_2) f_2(E_2)(1 - f_3(E_3)) \tag{15.535}$$

Therefore, the net capture rate is [133]:

$$C = R - M = (1 - e^{\eta_2 - \eta_3})\left(1 - \frac{n_2}{N_2}\right)\frac{n_3}{\tau} \tag{15.536}$$

where:

$$N_2 = \int_{E_W}^{E_c} dE_2 g_2(E_2) \tag{15.537}$$

$\eta_2 = (-q\Phi_2 - E_C)/k_B T$ and $\eta_3 = (-q\Phi_3 - E_C)/k_B T$ contain the quasi-Fermi level information and τ is the capture time. The capture time is representative of the carrier–carrier and carrier-optical phonon scattering processes when the carriers are scattered into the quantum well.

This capture rate is added to the continuity equations as a recombination term. The treatment for holes is similar. It is apparent that separate capture times must be given for the electrons and holes accordingly, and these can be specified by the keywords `QWeScatTime` and `QWhScatTime` in the `Physics-Laser` section of the command file.

For shallow quantum wells, the energy transfer during scattering can only occur in a limited range. In the limit of elastic scattering, the net capture rate for shallow quantum wells is:

$$C = \left(\frac{F_{3/2}(\eta_3)}{F_{1/2}(\eta_3)} - \frac{F_{3/2}(\eta_2)}{F_{1/2}(\eta_3)}\right)\frac{\eta_3}{\tau} \tag{15.538}$$

where F_m is the Fermi integral of the order of m. The shallow quantum well model is especially suitable for InGaAsP-type quantum wells that have smaller conduction and valence band offsets compared to InGaAs and GaAs wells. It is activated by the keyword `QWShallow`.

The activating syntax for the QW scattering model is located in the `Physics-Laser` and `Solve` sections of the command file:

```
Physics {...
   Laser (...
      Optics (...)
      # ---- Use 3-point QW model ----
      QWTransport
      QWExtension = AutoDetect
      # ---- Activate QW scattering model ----
      QWScatModel
      QWeScatTime = 8e-13       # [s]
      QWhScatTime = 4e-13       # [s]
      eQWMobility = 1450        # [cm^2/Vs]
      hQWMobility = 370         # [cm^2/Vs]
#     QWShallow                 # for shallow QWs only
   )
}
...
Solve {
   Coupled {Poisson}
   Coupled {Poisson Electron Hole}
   Coupled {Poisson Electron Hole QWeScatter QWhScatter}
   Coupled {Poisson Electron Hole QWeScatter QWhScatter PhotonRate}
...
}
```

Some comments about the syntax are:

- Switching on the QW scattering model with `QWScatModel` requires that the keyword `QWTransport` is included to set up the ‘three-point’ QW model.

- The keyword `QWExtension=AutoDetect` activates the automatic extraction of quantum-well thickness according to the keyword `Active` in the material region `Physics` section. This is the default behavior. If all quantum wells have the same thickness, the extracted values can be overridden by specifying `QWExtension=<float>` (in nanometers).
- `QWShallow` should only be activated for shallow quantum wells.
- In the `Solve` section, equation systems are added consecutively. Each `Coupled` solution provides the initial guesses for the next `Coupled` problem containing more equation systems. In this case, `Coupled {Poisson Electron Hole}` solves the quantum well transport with thermionic emission only. The solution provides a good initial guess for the next `Coupled` problem when the QW scattering models are added.
- The QW scattering model creates the continuity equations for the bound carriers, so the user must input the QW bound carrier mobilities with the keywords `eQWMobility` and `hQWMobility`.

28.3 Radiative recombination and gain coefficients

After the carriers are captured in the active region, they experience either dark recombination processes (such as Auger and SRH) or radiative recombination processes (such as stimulated and spontaneous emissions), or escape from the active region. This section describes how stimulated and spontaneous emissions are computed in DESSIS.

28.3.1 Stimulated and spontaneous emission coefficients

In the active region of the laser, radiative recombination is treated locally at each active vertex. The stimulated and spontaneous emissions are computed using Fermi's golden rule. At each active vertex of the quantum wells, the local stimulated emission coefficient is:

$$r^{st}(\hbar\omega) = \sum_{i,j}\int dEC_o k_{st}|M_{i,j}|^2 D(E) \times (f_i^C(E) + f_j^V(E) - 1)L(E) \tag{15.539}$$

and the local spontaneous emission coefficient is:

$$r^{sp}(\hbar\omega) = \sum_{i,j}\int dEC_o k_{sp}|M_{i,j}|^2 D(E) f_i^C(E) f_j^V(E) L(E) \tag{15.540}$$

where:

$$C_0 = \frac{\pi e^2}{n_g c \varepsilon_0 m_0^2 \omega} \tag{15.541}$$

$$|M_{i,j}|^2 = P_{ij}|O_{i,j}|^2\left(\frac{m_0}{m_e} - 1\right)\frac{m_0 E_g(E_g + \Delta)}{12\left(E_g + \frac{2}{3}\Delta\right)} \tag{15.542}$$

$$O_{i,j} = \int_{-\infty}^{\infty} dx\zeta_i(x)\zeta^*{}_j(x) \tag{15.543}$$

$$f^C_{(i,\Phi_n,E)} = \left(1 + \exp\left(\frac{E_C + E_i + q\Phi_n + \frac{m_r}{m_e}E}{k_B T}\right)\right)^{-1} \tag{15.544}$$

$$f^V_{(j,\Phi_p,E)} = \left(1 + \exp\left(\frac{E_V - E_j + q\Phi_p - \frac{m_r}{m_h}E}{k_B T}\right)\right)^{-1} \tag{15.545}$$

$$D^r_{(E)} = \frac{m_r}{\pi\hbar^2 L_x} \tag{15.546}$$

$$m_r = \left(\frac{1}{m_e} + \frac{1}{m_h}\right)^{-1} \tag{15.547}$$

$L(E)$ is the gain-broadening function. The electron, light-hole, and heavy-hole subbands are denoted by the indices i and j. $f^C_{i(E)}$ and $f^V_{j(E)}$ are the local Fermi–Dirac distributions for the conduction and valence bands, $D^r_{(E)}$ is the reduced density of states, $|O_{i,j}|^2$ is the overlap integral of the quantum mechanical wavefunctions, and P_{ij} is the polarization-dependent factor of the momentum (optical) matrix element $|M_{i,j}|^2$. The spin-orbit split-off energy is Δ and E_g is the band-gap energy. The anisotropic polarization-dependent factor P_{ij} in the optical matrix element is discussed in Section 28.8 on page 15.453. These emission coefficients determine the rate of production of photons when given the number of available quantum well carriers at the active vertex.

k_{st} and k_{sp} are scaling factors for the optical matrix element $|M_{i,j}|^2$ of the stimulated and spontaneous emissions, respectively. They have been introduced to allow users to tune the stimulated and spontaneous gain curves. Consequently, these parameters can change the threshold current. The activating keywords are `StimScaling` and `SponScaling` in the `Physics-Laser` section of the command file:

```
Physics {...
   Laser (...
      Optics (...)
      # ---- Scale stimulated & spontaneous gain ----
      StimScaling = 1.0         # default value is 1.0
      SponScaling = 1.0         # default value is 1.0
   )
}
```

28.3.2 Active bulk material gain

The stimulated and spontaneous emission coefficients discussed are derived for the quantum well. However, these coefficients can apply to bulk materials with slight modifications. In bulk active materials, it is assumed that the optical matrix element is isotropic. The sum over the subbands is reduced to one electron, and one heavy-hole and one light-hole level, because there is no quantum-mechanical confinement in bulk material. In addition, the subband energies are set to $E_i = 0$, and the following coefficients are modified:

$$O_{i,j} = 1 \tag{15.548}$$

$$D^r(E) = \frac{1}{2\pi^2}\left(\frac{2m_r}{\hbar^2}\right)^{3/2} E^{1/2} \tag{15.549}$$

$$P_{i,j} = 1 \tag{15.550}$$

All other expressions remain the same.

28.3.3 Stimulated recombination rate

The radiative emissions contribute to the production of photons but they also deplete the carrier population in the active region. At each active vertex, the stimulated recombination rate of the carriers must be equal to the sum of the photon production rate of every lasing mode so that conservation of particles is ensured. The stimulated recombination rate for each active vertex is:

$$R^{st}(x,y) = \sum_i r^{st}(\hbar\omega_i) S_i |\Psi_i(x,y)|^2 \tag{15.551}$$

where the sum is taken over all lasing modes. The stimulated emission coefficient is computed locally at this active vertex and its value is taken at the lasing energy, $\hbar\omega_i$, of mode i. S_i is the photon rate of mode i, solved from the corresponding photon rate equation of mode i, and $|\Psi_i(x,y)|^2$ is the local optical field intensity of mode i at this active vertex. This stimulated recombination rate is entered in the continuity equations to account for the correct depletion of carriers by stimulated emissions.

28.3.4 Spontaneous recombination rate

Spontaneous emissions also deplete the carrier population in the active region and must be taken into consideration. The spontaneous recombination rate at each active vertex is:

$$R^{sp}_{tot}(x,y,z) = \int_0^\infty r^{sp}(E)\rho^{opt}(E)dE \tag{15.552}$$

where the optical mode density is:

$$\rho^{opt}(E) = \frac{n_g^2 E^2}{\pi^2 \hbar^3 c^2} \tag{15.553}$$

The spontaneous recombination rate is an integral over energy space, and this rate is entered into the carrier continuity equations.

28.3.5 Spontaneous emission power for LEDs

The total spontaneous emission power density at each active vertex (units of Js^{-1}m^{-3}) is:

$$\Delta P^{sp}(x,y,z) = \int_0^\infty r^{sp}(E)\rho^{opt}(E)\cdot(\hbar\omega)dE \tag{15.554}$$

where $\rho^{opt}(E)$ has been defined in (Eq. 15.553). This equation is similar to (Eq. 15.552), except that an additional energy term, $E = \hbar\omega$, is included in the integrand to account for the energy spectrum of the spontaneous emission. The total spontaneous emission power is simply the volume integral of the power density over all the active vertices:

$$P_{total}{}^{sp} = \int_{(active-region)} \Delta P^{sp}(x, y, z)dV \tag{15.555}$$

This is the total spontaneous emission power that is computed and output in an LED simulation.

28.4 Gain-broadening models

Three different line-shape broadening models are available: Lorentzian, Landsberg, and hyperbolic-cosine. These line-shape functions, $L(E)$, are embedded in the radiative emission coefficients in (Eq. 15.539) and (Eq. 15.540) to account for broadening of the gain spectrum.

28.4.1 Lorentzian broadening

Lorentzian broadening assumes that the probability of finding an electron or a hole in a given state decays exponentially in time [131]. The line-shape function is:

$$L(E) = \frac{\Gamma/(2\pi)}{(E_g - \hbar\omega + E)^2 + (\Gamma/2)^2} \tag{15.556}$$

28.4.2 Landsberg broadening

The Landsberg model gives a narrower, asymmetric line-shape broadening, and its line-shape function is:

$$L(E) = \frac{(\Gamma(E))/(2\pi)}{(E_g - \hbar\omega + E)^2 + (\Gamma(E)/2)^2} \tag{15.557}$$

where:

$$\Gamma(E) = \Gamma \sum_{k=0}^{3} a_k \left(\frac{E}{q\Psi_p - q\Psi_n}\right)^k \tag{15.558}$$

and $q\Psi_p - q\Psi_n$ is the quasi-Fermi level separation.

The coefficients a_k are:

$$\begin{aligned} a_0 &= 1 \\ a_1 &= -2.229 \\ a_2 &= 1.458 \\ a_3 &= -0.229 \end{aligned} \tag{15.559}$$

28.4.3 Hyperbolic-cosine broadening

The hyperbolic-cosine function has a broader tail on the low-energy side compared to Lorentzian broadening, and the line-shape function is:

$$L(E) = \frac{1}{4\Gamma} \cdot \frac{1}{\cosh^2\left(\frac{E}{2\Gamma}\right)} \tag{15.560}$$

28.4.4 Syntax to activate broadening

The user can select only one line-shape function for gain broadening. This is activated by the keyword `Broadening` in the `Physics-Laser` section of the command file:

```
Physics {...
   Laser (...
      Optics (...)
      # --- Lineshape broadening functions, choose one only ----
      Broadening (Type=Lorentzian Gamma=0.01)
#     Broadening (Type=Landsberg Gamma=0.01)     # Gamma in [eV]
#     Broadening (Type=CosHyper Gamma=0.01)
   )
}
```

`Gamma` is the line width Γ, which must be defined in units of eV. If no `Broadening` keyword is detected, DESSIS assumes the gain is unbroadened and does not perform the energy integral in (Eq. 15.539) and (Eq. 15.540).

28.5 Nonlinear gain saturation effects

Nonlinear gain saturation is caused by the interaction of increasing light intensity with the optical matrix element, and this effect is important in the study of modulation response. An infinite order perturbation approach is used in a density matrix formulation to derive the nonlinear gain effects [192]. Using this approach, the stimulated emission coefficient is derived to be [193]:

$$r^{st}(\hbar\omega) = \sum_{i,j}\int dEC_o k_{st}|M_{i,j}|^2 D(E) \times (f_i^C(E) + f_j^V(E) - 1)L(E) \tag{15.561}$$

where:

$$L(E) = \frac{\Gamma/(2\pi)}{(E-\hbar\omega)^2 + (\Gamma/2)^2 + \varepsilon S} \tag{15.562}$$

$$\varepsilon = \frac{2(\tau_c + \tau_v)|M_{i,j}|^2(\hbar\omega)\hbar^2}{\tau_{in} m_0^2 \varepsilon_0 n_g^2 E} \cdot |\Psi(x,y)|^2 \tag{15.563}$$

τ_c and τ_v are the intraband scattering times of the electrons and holes, respectively. Γ is the line width, which is defined by $\Gamma = 2(\hbar/\tau_{in})$. τ_{in} is the polarization relaxation time and can be defined as:

$$\frac{1}{\tau_{in}} = \frac{1}{2}\left(\frac{1}{\tau_c} + \frac{1}{\tau_v}\right) + \frac{1}{\tau_{sp}} \tag{15.564}$$

where τ_{sp} is the electron spontaneous emission lifetime and $|\Psi(x,y)|^2$ is the normalized local optical intensity. The stimulated emission coefficient with nonlinear gain effects in (Eq. 15.561) is of the same form as the original stimulated emission coefficient in (Eq. 15.539). The variables in the two coefficients are the same, except for the gain-broadening function, $L(E)$.

The nonlinear gain saturation effect can be included in the form of a broadening function. The simple gain saturation model commonly cited is $g = g_0/(1+\varepsilon S)$. This simple form gives a homogeneous broadening and can be approximated from (Eq. 15.561) by assuming $(\hbar/\tau_{in}) \gg (E-\hbar\omega)$. In this model, $|M_{i,j}|^2$ has no spectral dependence by definition.

Since the nonlinear gain saturation effect exists in the form of a broadening function, it can be activated by the keyword `Broadening` in the `Physics-Laser` section of the command file:

```
Physics {...
   Laser (...
      Optics (...)
         # ----- Gain saturation model -----
         Broadening(Type = LorentzianSat
                   Gamma = 4.39e-4             # [eV]
                   eIntrabandRelTime = 1e-13     # [s]
                   hIntrabandRelTime = 1e-13     # [s]
                   PolarizationRelTime = 3e-12   # [s]
         )
   )
}
```

Users must set the electron and hole intraband scattering times with `eIntrabandRelTime` and `hIntrabandRelTime`, respectively. The `PolarizationRelTime` can be computed from (Eq. 15.564).

NOTE If the nonlinear gain saturation is activated, users cannot select other gain-broadening functions.

28.6 Simple quantum well subband model

This section describes the solution of the Schrödinger equation for a simple finite quantum well model. This is the default model in DESSIS. A more advanced model using the k.p method is available (see Section 28.9 on page 15.455). This simple QW subband model is combined with separate QW strain (see Section 28.7 on page 15.451) and polarization dependence of the optical matrix element (see Section 28.8 on page 15.453) to model most quantum well systems.

In a quantum well, the carriers are confined in one direction. Of interest are the subband energies and wavefunctions of the bound states, which can be solved from the Schrödinger equation. In this simple QW subband model, it is assumed that the bands for the electron, heavy hole, and light hole are decoupled, and the subbands are solved independently by a 1D Schrödinger equation.

The time-independent 1D Schrödinger equation in the effective mass approximation is:

$$\left(-\frac{\hbar^2}{2}\frac{\partial}{\partial x}\frac{1}{m(x)}\frac{\partial}{\partial x}+V(x)-E^i\right)\zeta^i(x) = 0 \qquad (15.565)$$

where $\zeta^i(x)$ is the i-th quantum mechanical wavefunction, E^i is the i-th energy eigenvalue, and $V(x)$ is the finite well shape potential.

With the following ansatz for the even wavefunctions:

$$\zeta(x) = C_1 \begin{cases} \cos\left(\frac{\kappa l}{2}\right) e^{-\alpha(|x| - l/2)} & , \quad |x| > l/2 \\ \cos(\kappa x) & , \quad |x| \le l/2 \end{cases} \tag{15.566}$$

and the odd wavefunctions:

$$\xi(x) = C_2 \begin{cases} \pm\sin\left(\frac{\kappa l}{2}\right) e^{\mp(x \mp l/2)} & , |x| > l/2 \\ \sin(\kappa x) & , |x| \le l/2 \end{cases} \tag{15.567}$$

(Eq. 15.565) becomes [131]:

$$\alpha\frac{l}{2} + \frac{m_b}{m_w}\kappa\frac{l}{2}\cot\left(\kappa\frac{l}{2}\right) = 0 \tag{15.568}$$

$$\alpha\frac{l}{2} - \frac{m_b}{m_w}\kappa\frac{l}{2}\tan\left(\kappa\frac{l}{2}\right) = 0 \tag{15.569}$$

with:

$$\kappa = \frac{\sqrt{2m_w E}}{\hbar} \tag{15.570}$$

$$\alpha = \frac{\sqrt{2m_b(\Delta E_c - E)}}{\hbar} \tag{15.571}$$

The first transcendental equation gives the even eigenvalues, and the second one gives the odd eigenvalues. The wavefunctions are immediately obtained with (Eq. 15.566) and (Eq. 15.567) after the subband energy E has been computed. Having obtained the wavefunctions and subband energies, the carrier densities of the 1D-confined system are also computed by:

$$n(x) = N_e^{2D}\sum_i |\zeta_i(x)|^2 F_0(\eta_n - E_i) \tag{15.572}$$

$$p(x) = N_{hh}^{2D}\sum_j |\zeta_{j(x)}|^2 F_0(\eta_p - E_{hh}^j) + N_{lh}^{2D}\sum_m |\zeta_{m(x)}|^2 F_0(\eta_p - E_{lh}^m) \tag{15.573}$$

where $F_0(x)$ is the Fermi integral of the order 0, and η_n, η_p denote the chemical potentials. The indices *hh* and *lh* denote the heavy and light holes, respectively. The effective densities of states are:

$$N_e^{2D} = \frac{k_B T m_e}{\hbar^2 \pi L_x} \tag{15.574}$$

$$N_{lh/hh}^{2D} = \frac{k_B T m_{lh/hh}}{\hbar^2 \pi L_x} \tag{15.575}$$

where L_x is the thickness of the quantum well. The thickness of each quantum well is automatically detected in DESSIS by scanning the material regions for the keyword `Active`.

The effective masses of the carriers in the quantum well can be changed inside the parameter file:

```
eDOSMass
{
 * For effective mass specification Formula1 (me approximation):
 * or Formula2 (Nc300) can be used :
       Formula = 2     # [1]
 * Formula2:
 * me/m0 = (Nc300/2.540e19)^2/3
 * Nc(T) = Nc300 * (T/300)^3/2
       Nc300    = 8.7200e+16    # [cm-3]
 * Mole fraction dependent model.
 * If just above parameters are specified, then its values will be
 * used for any mole fraction instead of an interpolation below.
 * The linear interpolation is used on interval [0,1].
       Nc300(1)        = 6.4200e+17    # [cm-3]
}
...
SchroedingerParameters:
{ * For the hole masses for Schroedinger equation you can
 * use different formulas.
 * formula=1 (for materials with Si-like hole band structure)
 *   m(k)/m0=1/(A+-sqrt(B+C*((xy)^2+(yz)^2+(zx)^2)))
 *   where k=(x,y,z) is unit normal vector in reciprocal
 *   space. '+' for light hole band, '-' for heavy hole band
 * formula=2: Heavy hole mass mh and light hole mass ml are
 *   specified explicitly.
 * Formula 2 parameters:
       Formula = 2     # [1]
       ml      = 0.027 # [1]
       mh      = 0.08  # [1]
 * Mole fraction dependent model.
 * If just above parameters are specified, then its values will be
 * used for any mole fraction instead of an interpolation below.
 * The linear interpolation is used on interval [0,1].
       ml(1)   = 0.094 # [1]
       mh(1)   = 0.08  # [1]
}
```

28.6.1 Syntax for simple quantum well model

This simple QW subband model is the default model when the `QWTransport` and QW scattering model (`QWScatModel`) are activated. Table 15.154 summarizes the keywords that are associated with this simple QW model. Most keywords have been previously described except `Strain`, `SplitOff`, and `QWEffPeriod`. The keyword `Strain` is discussed in Section 28.7 on page 15.451.

Table 15.154 Keywords in Physics-Laser or Physics-LED sections associated with simple QW subband model

Feature keyword	Description
`eQWMobility=<float>`	2D mobility for bound electrons for scattering model [cm^2/Vs].
`hQWMobility=<float>`	2D mobility for bound holes for scattering model [cm^2/Vs].
`QWeScatTime=<float>`	Electron scattering time into quantum well [s].

Table 15.154 Keywords in Physics-Laser or Physics-LED sections associated with simple QW subband model

Feature keyword	Description
`QWExtension=AutoDetect`	Width of each quantum well automatically detected. The quantum well region must be specified by the keyword `Active`.
`QWhScatTime=<float>`	Hole scattering time into quantum well [s].
`QWScatModel`	Activates scattering model for quantum well.
`QWShallow`	Switches on scattering for flat quantum wells.
`QWTransport`	Activates the 'three-point' QW model with thermionic emission.
`Strain`	Activates the strain model for quantum well.

28.7 Strain effects

In semiconductor laser design, it is well known that strain of the quantum well modifies the laser characteristics. Due to the deformation potentials in the crystal at the well–bulk interface and valence band mixing effects, band structure modifications occur mainly for the valence bands. They have an impact on the optical recombination and transport properties. A more advanced k.p model is available, which incorporates strain into the Hamiltonian implicitly (see Section 28.9 on page 15.455).

In the simple QW subband model discussed in the previous section, a simpler approach to the QW strain effects is adopted. The simple QW subband model does not include nonparabolicities of the band structure, arising from valence band mixing and strain, in a rigorous manner. However, by carefully selecting the effective masses in the well, a good approximation of the strained band structure can be obtained [138]. The effective masses can be changed in the parameter file as previously shown.

Basically, strain has two impacts on the band structure. Due to the deformation potentials, the effective band offsets of the conduction and valence bands are modified. This is included in the simple QW subband model by:

$$\delta E_C = 2a_c\left(1-\frac{C_{12}}{C_{11}}\right)\varepsilon \tag{15.576}$$

$$\delta E_V^{HH} = 2a_v\left(1-\frac{C_{12}}{C_{11}}\right)\varepsilon + b\left(1+2\frac{C_{12}}{C_{11}}\right)\varepsilon \tag{15.577}$$

$$\delta E_V^{LH} = 2a_v\left(1-\frac{C_{12}}{C_{11}}\right)\varepsilon - b\left(1+2\frac{C_{12}}{C_{11}}\right)\varepsilon + \frac{\Delta}{2} - \left(-\frac{1}{2}\sqrt{\Delta^2+9\delta E_{sh}^2-2\delta E_{sh}\Delta}\right) \tag{15.578}$$

where a_n and a_c are the hydrostatic deformation potential of the conduction and valence bands, respectively. The shear deformation potential is denoted with b and Δ is the spin-orbit split-off energy. The elastic stiffness constants are C_{11} and C_{12}, and ε is the relative lattice constant difference in the active region. The hydrostatic component of the strain shifts the conduction band offset by δE_C and shifts the valence band offset by $\delta E_V^0 = 2a_v(1-C_{12}/C_{11})\varepsilon$.

The shear component of the strain decouples the light hole and heavy hole bands at the Γ point, and shifts the valence bands by an amount of $\delta E_{sh} = b(1+2C_{12}/C_{11})\varepsilon$ in opposite directions.

28.7.1 Syntax for quantum well strain

The strain shift can be activated by the keyword `Strain` in the `Physics-Laser` section of the command file:

```
Physics {...
   Laser (...
      Optics (...)
      # --- QW physics ---
      QWTransport
      QWExtension = AutoDetect
      QWScatModel
      QWeScatTime = 8e-13       # [s]
      QWhScatTime = 4e-13       # [s]
      eQWMobility = 5370        # [cm^2/Vs]
      hQWMobility = 150         # [cm^2/Vs]
      # --- QW strain ---
      Strain
   )
}
```

NOTE The value for the spin-orbit split-off energy is no longer defined in the command file. Enter it in the parameter file.

The parameters a_n, a_c, and b can be entered as `a_nu`, `a_c`, and `b_shear`, respectively, in the `QWStrain` section of the parameter file:

```
QWStrain
{
 * Deformation Potentials (a_nu, a_c, b, C_12, C_11
 * and strainConstant eps :
 * Formula:
 * eps = (a_bulk - a_active)/a_active
 * dE_c = ....
 * dE_lh = ...
 * dE_hh = ...
        eps     = -1.0000e-02         # [1]
        * a_nu  = 1.27                # [1]
        * a_c   = -5.0400e+00         # [1]
        * b_shear = -1.7000e+00       # [1]
        * C_11 = 10.11                # [1]
        * C_12 = 5.61                 # [1]
}
```

The elastic stiffness constants C_{11} and C_{12} can be specified by `C_11` and `C_12`. Due to valence band mixing and strain, the valence bands can become nonparabolic. However, within a small range from the band edge, parabolicity can still be assumed. In the simulation, users can modify the effective heavy hole and light hole masses in the parameter file for the subband calculation to account for this effect. The spin-orbit split-off energy can be specified in the `BandstructureParameters` section of the parameter file:

```
BandstructureParameters{
   ...
   so = 0.34        # [eV]
   ...
}
```

28.8 Polarization-dependent optical matrix element

The polarization dependence of the optical matrix element P_{ij} in (Eq. 15.542) can be treated in an elegant way for the simple QW subband model. Define the optical polarization angle, γ, as the angle of the vectorial optical field between the quantum well (QW) plane and perpendicular to the QW plane, as shown in Figure 15.108.

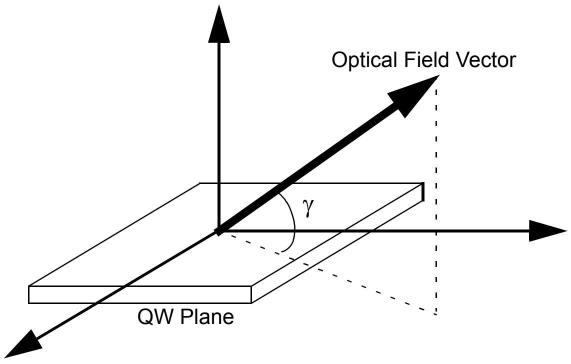


Figure 15.108 Taking the optical polarization with respect to QW plane so that anisotropic polarization-dependent factor in optical matrix element can be defined

If the vector lies completely in the QW plane as in the case of TE modes, the angle is 0. If it is strictly perpendicular as in TM modes, it is $\pi/2$. The unit is radian. These polarization angles at each vertex can be visualized by including the keywords `OpticalPolarizationAngleMode0` to `OpticalPolarizationAngleMode9` in the `Plot` section of the DESSIS command file.

NOTE In bulk material, the optical matrix element is isotropic and the polarization-dependent factor can be taken as $P_{ij} = 1$.

The polarization-dependent factor P_{ij} is a measure of the influence of the polarization of the optical field with the plane wavefunctions of the optical transition. Suppose the optical field polarization is defined by the unit vector:

$$\hat{e} = \alpha\hat{a}_{TE} + \beta\hat{a}_{TM} \tag{15.579}$$

where $\alpha^2 + \beta^2 = 1$. It is obvious that $\alpha = \cos\gamma$ and $\beta = \sin\gamma$ from Figure 15.108. The resultant polarization-dependent factor P_{ij} for a quantum well can be derived as:

$$\begin{aligned} P_{c,hh} &= \frac{1}{M_b^2}\left|\hat{e} \bullet \langle iS\mathrm{i}'|\bar{p}|\frac{3}{2}, -\frac{3}{2}\rangle'\right|^2 \\ &= 3\left\{\frac{1}{4}\alpha^2 cos^2\Psi + \frac{1}{4}\alpha^2 + \frac{1}{2}\beta^2 sin^2\Psi\right\} \end{aligned} \tag{15.580}$$

for C-HH transitions, and:

$$P_{c,lh} = \frac{1}{M_b^2}\left\{\left|\hat{e} \bullet \langle iSi'|\bar{p}|\frac{3}{2}, -\frac{1}{2}\rangle'\right|^2 + \left|\hat{e} \bullet \langle iSi'|\bar{p}|\frac{3}{2}, \frac{1}{2}\rangle'\right|^2\right\} \tag{15.581}$$

$$= 3\left\{\frac{1}{3}\alpha^2 sin^2\Psi + \frac{1}{12}\alpha^2 cos^2\Psi + \frac{1}{12}\alpha^2 + \frac{2}{3}\beta^2 cos^2\Psi + \frac{1}{6}\beta^2 sin^2\Psi\right\}$$

for C-LH transitions. The **k**-wavevector angle is defined as:

$$cos^2\Psi = \frac{E_g + E_i + E_j}{E}, E > E_g + E_i + E_j \tag{15.582}$$

The cross-coupling terms between the TE and TM contributions evaluate to zero upon integration over the polar angle in the QW plane. This is equivalent to computing different TE-stimulated and TM-stimulated emission coefficients and, therefore, different modal gains for the TE and TM polarizations, which are subsequently summed to give the overall modal gain of the mode at each active vertex, that is:

$$G_{modal}(x, y) = r_{TE}^{st}|\bar{E}_{TE}(x, y)|^2 + r_{TM}^{st}|\bar{E}_{TM}(x, y)|^2 \tag{15.583}$$

where $|\bar{E}_{TE}(x, y)|^2$ and $|\bar{E}_{TM}(x, y)|^2$ are the local normalized intensities of the vectorial optical field projected onto the TE and TM planes, respectively. In this way, automatic polarization-dependent gain calculations for each separate vertex in the active region are achieved if the vectorial optical solver (`FEVectorial`) is chosen.

In the case of scalar optical solvers (`FEScalar`), the user can select the type of polarization (TE or TM) in the `Physics-Laser-Optics-FEScalar` section of the command file:

```
Physics {...
   Laser (...
      Optics(
         FEScalar(EquationType = Waveguide
            Symmetry = Symmetric
            LasingWavelength = 800              # [nm]
            TargetEffectiveIndex = (3.4 3.34)   # initial guess
            TargetLoss = (10.0 12.0)            # initial guess [1/cm]
            # --- Specify the optical polarization ---
            Polarization = (TE TM)
            ModeNumber = 2
         )
      )
   )
}
```

The TE and TM polarizations give a global optical polarization angle of 0 and $\pi/2$, respectively. The default is TE polarization. For purely TE or TM polarizations, (Eq. 15.580) and (Eq. 15.581) reduce to the values in Table 15.155.

Table 15.155 Polarization-dependent factor for purely TE and TM polarizations

	TE polarization	**TM polarization**
C-HH	$\frac{3}{4}(1 + \cos^2\Psi)$	$\frac{3}{2}\sin^2\Psi$
C-LH	$\frac{5}{4} - \frac{3}{4}\cos^2\Psi$	$\frac{1}{2}(1 + 3\cos^2\Psi)$

A self-consistent simulation of four vectorial modes is performed for an edge-emitting laser to show the polarization-dependent effects on the gain. Figure 15.109 shows the modal gain plotted (using INSPECT) as a function of transition energy for the four different modes.

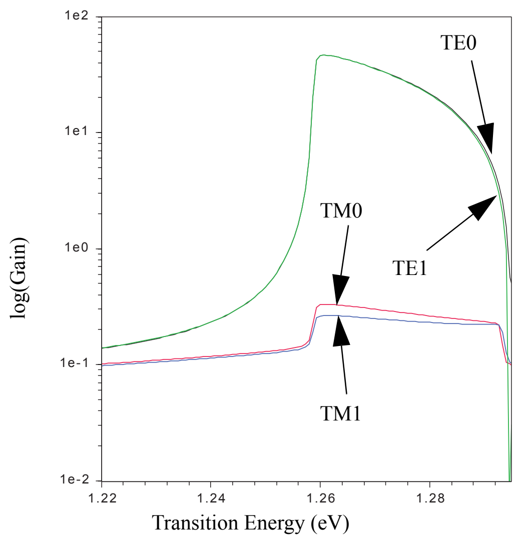


Figure 15.109 Modal gain curves for two TE and two TM modes

The four modes contain two quasi-TE (TE0 and TE1) and two quasi-TM (TM0 and TM2) modes. The TE gains are much greater than the TM gains because the InGaAs quantum wells used in this example strongly favor TE polarization. Therefore, DESSIS can simulate multimode lasing with mixed optical polarization.

28.9 k.p method

A more advanced model for computing the QW subbands, strain shift, many-body effects, and so on with the k.p method is available in DESSIS. The k.p method implemented in DESSIS is based on the Luttinger–Kohn model. Choices of 4×4, 6×6, and 8×8 Hamiltonians are possible to handle degenerate heavy hole, light hole, spin-orbit split-off, and conduction bands.

The wavefunction in a periodic lattice can be expanded into plane waveforms and is assumed to be a Bloch function:

$$\psi_{n\boldsymbol{k}}(\boldsymbol{r}) = e^{i\boldsymbol{k}\cdot\boldsymbol{r}} \cdot u_{n\boldsymbol{k}}(\boldsymbol{r}) \quad (15.584)$$

where $u_{n\boldsymbol{k}}(\boldsymbol{r})$ is the envelope wavefunction, which is periodic across each crystal lattice, $u_{n\boldsymbol{k}}(\boldsymbol{r}) = u_{n\boldsymbol{k}}(\boldsymbol{r}+\boldsymbol{R})$. Substituting this into the Schrödinger equation, $\boldsymbol{H}\psi_{n\boldsymbol{k}}(\boldsymbol{r}) = E_n(\boldsymbol{k})\psi_{n\boldsymbol{k}}(\boldsymbol{r})$ gives the eigenvalue equation for the envelope wavefunction:

$$\boldsymbol{H}' u_{n\boldsymbol{k}}(\boldsymbol{r}) = E'(\boldsymbol{k}) u_{n\boldsymbol{k}}(\boldsymbol{r}) \quad (15.585)$$

where the Luttinger–Kohn Hamiltonian is:

$$\boldsymbol{H}' = \frac{p^2}{2m} + V(\boldsymbol{r}) + \frac{\hbar^2 k^2}{2m_0} + \frac{\hbar}{4m_0^2 c^2}\nabla V \times \boldsymbol{p} \cdot \sigma + \frac{\hbar}{m_0}\mathbf{k} \cdot \boldsymbol{p} \quad (15.586)$$

σ is the Pauli spin matrix and V is the potential variation of the quantum well region, which may include carrier interaction effects.

In DESSIS, two types of carrier interaction effect are treated:

- Free carrier theory (FCT), which assumes that the carriers do not interact with each other and, therefore, do not contribute to any additional potential terms.
- Many-body effects, which account for the Coulombic interaction between the carriers. This is included in the form of a screened Hartree–Fock (SHF) potential.

When the k.p method is used, the valence band mixing effects and many-body effects (if included) distort the band structure from a parabolic shape. In this case, the integral in the stimulated and spontaneous emission coefficients ((Eq. 15.539) and (Eq. 15.540)) is performed in **k**-space rather than energy space.

The features of the k.p method are:

- Only Lorentzian and hyperbolic-cosine gain broadenings are available.
- Only scalar optical modes in single-mode simulation.
- For zinc-blende crystal structure, users can select the 4×4, 6×6, and 8×8 k.p Hamiltonians with the FCT and SHF carrier interaction effects.
- For wurtzite crystal structure, users can select only the 6×6 k.p Hamiltonian with the FCT and SHF carrier interaction effects.

NOTE The next release will enable multiple and vectorial transverse optical modes and the 8 x 8 Hamiltonian for the wurtzite crystal structure. An additional second Born approximation option to treat the many-body effect will also be introduced.

28.9.1 Luttinger–Kohn parameters and Hamiltonians for zinc-blende crystal structure

The Luttinger–Kohn parameters γ_1, γ_2, and γ_3 are well known. They are used in the following expressions, which are key components for forming the Luttinger–Kohn Hamiltonian matrices:

$$P = \left(\frac{\hbar^2\gamma_1}{2m}\right)(k_x^2 + k_y^2 + k_z^2) \tag{15.587}$$

$$Q = \left(\frac{\hbar^2\gamma_2}{2m}\right)(k_x^2 + k_y^2 - 2k_z^2) \tag{15.588}$$

$$R = -\left(\frac{\hbar^2\gamma_2}{2m}\right)\sqrt{3}(k_x^2 - k_y^2) + i\left(\frac{\hbar^2\gamma_3}{2m}\right)2\sqrt{3}k_xk_y \tag{15.589}$$

$$S = \left(\frac{\hbar^2\gamma_3}{2m}\right)2\sqrt{3}(k_x - ik_y)k_z \tag{15.590}$$

The strain-related parameters are:

$$\varepsilon_{xx} = \varepsilon_{yy} = \frac{a_{0b} - a_{0w}}{a_{0w}} \tag{15.591}$$

$$\varepsilon_{zz} = -\frac{2C_{12}}{C_{11}} \tag{15.592}$$

$$P_\varepsilon = a_v(\varepsilon_{xx} + \varepsilon_{yy} + \varepsilon_{zz}) = -\delta E_v \tag{15.593}$$

$$Q_\varepsilon = -\frac{b}{2}(\varepsilon_{xx} + \varepsilon_{yy} - 2\varepsilon_{zz}) \tag{15.594}$$

$$\delta E_c = 2a_c\left(1 - \frac{C_{12}}{C_{11}}\right)\varepsilon_{xx} \tag{15.595}$$

where a_{0w} and a_{0b} are the lattice constants of the well and barrier, respectively. C_{11}, C_{12}, a_v, and b are deformation potentials. As shorthand notations, the following are defined:

$$P_t = P + P_\varepsilon \tag{15.596}$$

$$Q_t = Q + Q_\varepsilon \tag{15.597}$$

The envelope wavefunction, $u_{n\boldsymbol{k}}$, is also expanded to a set of basis functions: p-like for the valence bands and s-like for the conduction bands. Valence band mixing is also included in this expansion.

Four-band Hamiltonian

The 4×4 Hamiltonian gives the solution to degenerate heavy-hole and light-hole bands; the conduction band is treated independently. Its matrix form is:

$$\boldsymbol{H'} = \begin{bmatrix} P_t + Q_t & -S & R & 0 \\ -S^* & P_t - Q_t & 0 & R \\ R^* & 0 & P_t - Q_t & S \\ 0 & R^* & S^* & P_t + Q_t \end{bmatrix} \tag{15.598}$$

where * denotes complex conjugate.

Six-band Hamiltonian

The 6×6 Hamiltonian gives the solution to degenerate heavy-hole, light-hole, and split-off bands; the conduction band is treated independently. Its matrix form is:

$$\boldsymbol{H}' = \begin{bmatrix} P_t + Q_t & R & \sqrt{2}R & -\frac{S}{\sqrt{2}} & -S & 0 \\ R^* & P_t - Q_t & \sqrt{2}Q & \sqrt{\frac{3}{2}}S^* & 0 & S \\ \sqrt{2}R^* & \sqrt{2}Q & P_t + \Delta & 0 & \sqrt{\frac{3}{2}}S^* & -\frac{S}{\sqrt{2}} \\ -\frac{S^*}{\sqrt{2}} & \sqrt{\frac{3}{2}}S & 0 & P_t + \Delta & -\sqrt{2}Q & -\sqrt{2}R \\ -S^* & 0 & \sqrt{\frac{3}{2}}S & -\sqrt{2}Q & P_t - Q_t & R \\ 0 & S^* & -\frac{S^*}{\sqrt{2}} & -\sqrt{2}R^* & R^* & P_t + Q_t \end{bmatrix} \tag{15.599}$$

where Δ is the spin-orbit split-off energy.

Eight-band Hamiltonian

The 8×8 Hamiltonian gives the solution to degenerate heavy-hole, light-hole, split-off, and conduction bands. Its matrix form is:

$$\boldsymbol{H}' = \begin{bmatrix} E_g + s\frac{\hbar^2 k^2}{2m_0} + \delta E_c & -2P_z k_z & P_z k_z & \sqrt{\frac{3}{2}}P_+ & 0 & -\frac{P_-}{\sqrt{2}} & -P_- & 0 \\ -\sqrt{2}P_z^* k_z & -(P_t - Q_t) & -\sqrt{2}Q_t & S^* & -\frac{P_+^*}{\sqrt{2}} & 0 & -\sqrt{\frac{3}{2}}S & R \\ P_z^* k_z & -\sqrt{2}Q_t & -(P_t + \Delta) & -\frac{S^*}{\sqrt{2}} & -P_+^* & \sqrt{\frac{3}{2}}S & 0 & \sqrt{2}R \\ \sqrt{\frac{3}{2}}P_+^* & S & -\frac{S}{\sqrt{2}} & -(P_t + Q_t) & 0 & -R & -\sqrt{2}R & 0 \\ 0 & -\frac{P_+}{\sqrt{2}} & -P_+ & 0 & E_g + s\frac{\hbar^2 k^2}{2m_0} + \delta E_c & \sqrt{2}P_z k_z & -P_z k_z & -\sqrt{\frac{3}{2}}P_- \\ -\frac{P_-^*}{\sqrt{2}} & 0 & \sqrt{\frac{3}{2}}S^* & -R^* & \sqrt{2}P_z^* k_z & -(P_t - Q_t) & -\sqrt{2}Q_t & S \\ -P_-^* & -\sqrt{\frac{3}{2}}S^* & 0 & -\sqrt{2}R^* & -P_z^* k_z & -\sqrt{2}Q_t & -(P_t - \Delta) & -\frac{S}{\sqrt{2}} \\ 0 & R^* & \sqrt{2}R^* & 0 & -\sqrt{\frac{3}{2}}P_-^* & S^* & -\frac{S^*}{\sqrt{2}} & -(P_t + Q_t) \end{bmatrix} \tag{15.600}$$

where:

$$P_z = i\frac{P}{\sqrt{3}} \tag{15.601}$$

$$P_{\pm} = i\frac{P}{\sqrt{3}}(k_x \pm k_y) \tag{15.602}$$

and s is a dimensionless parameter for describing the effect of the free-electron term in the Kane approach [194].

28.9.2 Luttinger–Kohn parameters and Hamiltonian for wurtzite crystal structure

Six-band Hamiltonian

The 6×6 Hamiltonian gives the solution to degenerate heavy-hole, light-hole, and split-off bands; the conduction band is treated independently. The formula from the paper by Chuang is followed [198]. Its block-diagonalized matrix form is:

$$\boldsymbol{H}' = \begin{bmatrix} F & K_t & -iH_t & 0 & 0 & 0 \\ K_t & G & \Delta - iH_t & 0 & 0 & 0 \\ iH_t & \Delta + iH_t & \lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & F & K_t & iH_t \\ 0 & 0 & 0 & K_t & G & \Delta + iH_t \\ 0 & 0 & 0 & -iH_t & \Delta - iH_t & \lambda \end{bmatrix} \tag{15.603}$$

where:

$$F = \Delta_1 + \Delta_2 + \lambda + \theta \tag{15.604}$$

$$F = \Delta_1 - \Delta_2 + \lambda + \theta \tag{15.605}$$

$$\lambda = \frac{\bar{h}^2}{2m_0}(A_1 k_z^2 + A_2 k_t^2) + \lambda_\varepsilon \tag{15.606}$$

$$\lambda_\varepsilon = D_1 \varepsilon_{zz} + D_2(\varepsilon_{xx} + \varepsilon_{yy}) \tag{15.607}$$

$$\theta = \frac{\bar{h}^2}{2m_0}(A_3 k_z^2 + A_4 k_t^2) + \theta_\varepsilon \tag{15.608}$$

$$\theta_\varepsilon = D_3 \varepsilon_{zz} + D_4(\varepsilon_{xx} + \varepsilon_{yy}) \tag{15.609}$$

$$K_t = \frac{\bar{h}^2}{2m_0} A_5 k_t^2 \tag{15.610}$$

$$H_t = \frac{\hbar^2}{2m_0} A_6 k_z k_t \tag{15.611}$$

28.9.3 Syntax for k.p method

The syntax to activate the k.p method is complicated. It involves:

- The definition of a nonlocal mesh across the quantum wells so that a 1D (spatially) Schrödinger equation can be formulated with the k.p method.
- Using new keywords in the `Physics-Laser` and `Solve` sections.
- Inputting new coefficients in the `QWStrain` section of the parameter file.

Constructing a nonlocal mesh on a straight line

The user must include an additional layer between the separate confinement heterostructure (SCH) and quantum well regions, each on the n and p side of the laser diode, as shown in Figure 15.110. These additional layers together with the quantum well region define the spatial span where the 1D Schrödinger equation will be solved numerically using the k.p method. The 1D Schrödinger equation is solved by discretizing the line (nonlocal mesh) that traverses these additional layers and the quantum well region in the vertical direction. The user can select the discretization size in the command file.

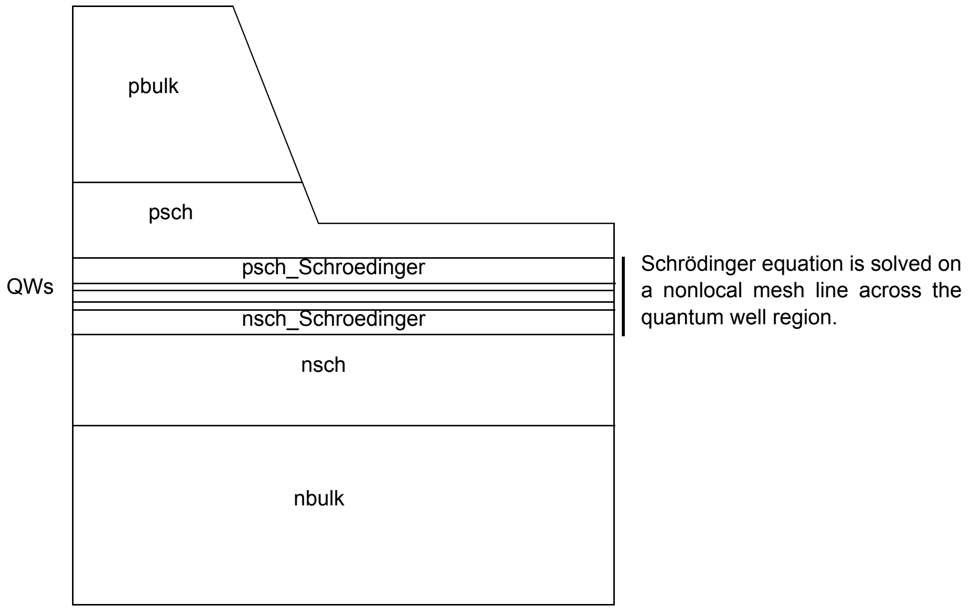

pbulk
psch
psch_Schroedinger
QWs
nsch_Schroedinger
nsch
nbulk
Schrödinger equation is solved on a nonlocal mesh line across the quantum well region.

Figure 15.110 Adding additional layers between the SCH and quantum well regions; these layers are labelled psch_Schroedinger and nsch_Schroedinger as they are associated with the solution of 1D Schrödinger equation

The k.p method is activated by the keyword `BandStructure` in the `Physics-Laser` and `Solve` sections of the command file:

```
Electrode {
    {Name="p_Contact" voltage=0.9 AreaFactor = 500}
    {Name="n_Contact" voltage=0.0 AreaFactor = 500}
}
```

```
File {
   Grid = "mesh_mdr.grd"
   Doping = "mesh_mdr.dat"
   Parameters = "des_las.par"

   Current = "testkp_current"
   Output = "testkp_log"
   Plot = "testkp_plot"
   ModeGain= "testkp_gain"
}

Plot {
   # Include desired variables to plot
}

Physics {
   AreaFactor = 2       # for symmetric devices
   Laser (
      Optics(
         # --- Only scalar solver possible with k.p method in this release ---
         FEScalar( Symmetry = Symmetric
                   Polarization = TE            # or TM
                   LasingWavelength = 980       # [nm]
                   TargetEffectiveIndex = 3.5   # initial guess
                   ModeNumber = 1
         )
      )
      TransverseModes                           # Edge emitter
      CavityLength = 500                        # [microns]
      lFacetReflectivity = 0.3
      rFacetReflectivity = 0.3
      OpticalLoss = 10.0                        # [1/cm]

      # --- Define basic QW physics and three-point model ---
      QWTransport
      QWExtension = AutoDetect
      QWScatmodel
      QWeScatTime = 1e-13                       # [s]
      QWhScatTime = 2e-14                       # [s]
      eQWMobility = 9200                        # [cm^2/Vs]
      hQWMobility = 400                         # [cm^2/Vs]

      # --- Activate k.p method with strain effects ---
      ManyBodyEffects (Type=FCT)                 # or SHF
      Strain (RefLatticeConst = 5.65392e-10)     # [m]
      Bandstructure (
         CrystalType = Zincblende       # Wurtzite in next release
         Order = kp4x4                  # or Nokp or kp6x6 or kp8x8
          )
      # --- Only Lorentzian or CosHyper gain broadening with k.p method ---
      Broadening(Type=Lorentzian Gamma=0.013)

   )
   Mobility (DopingDep)
   Recombination (SRH Auger)
   EffectiveIntrinsicDensity (NoBandGapNarrowing)
   Fermi
   HeteroInterfaces
   Thermionic
}

# --- Define material region physics ---
```

```
Physics (region="pbulk") { MoleFraction(xfraction = 0.28) }
Physics (region="nbulk") { MoleFraction(xfraction = 0.28) }
Physics (region="psch") { MoleFraction(xfraction = 0.09) }
Physics (region="nsch") { MoleFraction(xfraction = 0.09) }
Physics (region="psch_Schroedinger") { MoleFraction(xfraction = 0.09) }
Physics (region="nsch_Schroedinger") { MoleFraction(xfraction = 0.09) }
Physics (region="barr1") { MoleFraction(xfraction = 0.09) }

# --- Define the QWs as the active region ---
Physics (region="qw1") {
   MoleFraction( xfraction = 0.8 Grading = 0.00 )
   Active
}
Physics (region="qw2") {
   MoleFraction( xfraction = 0.8 Grading = 0.00 )
   Active
}

# --- Activate Schroedinger solver on non-local mesh defined below ---
Physics (RegionInterface="psch/psch_Schroedinger") {
   Schroedinger (electron maxSolutions(electron)=3
                 hole maxSolutions(hole)=6
   )
}

# --- Define non-local mesh ---
Math (RegionInterface="psch/psch_Schroedinger") {
   Nonlocal( Length = 0.143e-4          # [cm]
             Direction = 2              # y-direction
             Discretization = 0.2e-7    # [cm]
   )
}
Math (RegionInterface="nsch/nsch_Schroedinger") {
   NonLocal(AnchorPoints)
}

# --- Define regions to be excluded from non-local mesh ---
Math (Region="nsch") {
   NonLocal(-Transparent)
}
Math (Region="psch") {
   NonLocal(-Transparent)
}
Math (Region="pbulk") {
   NonLocal(-Transparent)
}

# ----- Choice and control of numerical methods -----
Math {...}

Solve {
   Poisson
   Coupled {Electron Hole Poisson QWeScatter QWhScatter}
   Coupled {Electron Hole Poisson QWeScatter QWhScatter PhotonRate}

   # --- Use simple QW subband model first to solve ---
   quasistationary (
        InitialStep=0.05
        MaxStep=0.1
        Minstep=0.00001
        Goal {name="p_Contact" voltage=1.2})
        {
           Plugin(BreakOnFailure) {
              Wavelength
              Coupled {Electron Hole Poisson QWeScatter QWhScatter
                       PhotonRate}
           }
        }

   # --- Then turn on the k.p method ---
   Bandstructure
   Wavelength

   # --- Continue using the k.p method in a Gummel iteration for self-consistency ---
   quasistationary (
        InitialStep=0.05
        MaxStep=0.1
        Minstep=0.00001
        # ---
        Plot {range=(0,1) intervals=5}
        PlotGainPara {range=(1.0 1.2) intervals=60}
        PlotGain {range=(0,1) intervals=2 }
        Goal {name="p_Contact" voltage=2.5})
        {
           Plugin(BreakOnFailure) {
              Bandstructure
              Wavelength
              Coupled{Electron Hole Poisson QWeScatter QWhScatter
                      PhotonRate}
           }
        }
}
```

The material regions defined in this example correspond to those in Figure 15.110 on page 15.460. Some comments about this example are:

- In the `Physics-Laser` section, the k.p method with various effects is activated by the keywords `Bandstructure`, `Strain`, and `ManyBodyEffects`.
- In the `Physics-Laser-Strain` section, a reference lattice constant must be specified by the argument `RefLatticeConstant=<float>` to compute the strain parameters.
- In the `Physics-Laser-ManyBodyEffects` section, only two types of many-body effects are available: free carrier theory (keyword `FCT`) and screened Hartree–Fock (keyword `SHF`). The default is the free carrier theory.
- In the `Physics-Schroedinger` section, the 1D Schrödinger equation is activated to be solved on a nonlocal mesh defined by the boundaries of the `psch_Schroedinger` and `nsch_Schroedinger` layers (see Figure 15.110).
- The nonlocal mesh for the 1D Schrödinger equation is defined in the `Math-Nonlocal` sections. The `AnchorPoints` define the starting location of the 1D nonlocal mesh. The length and discretization size of the mesh can be defined by the keywords `Length` and `Discretization`, respectively. The mesh should only be within the `psch_Schroedinger`, `nsch_Schroedinger`, and quantum well regions only. Therefore, the value that is specified by the keyword `Length` should be approximately equal to the distance between the two interfaces, for which `Length` and `AnchorPoints` have been specified. If the length of the nonlocal mesh exceeds the boundaries, the user can manually exclude the unwanted material regions from the nonlocal mesh by using the keyword `NonLocal(-Transparent)`.
- In the `Solve` section, the simple QW subband model is solved up to near the lasing threshold. After that, the k.p method is included in the Gummel iterations for self-consistent solutions of the system.

Adjusting k.p parameters in parameter file

The deformation potentials and other strain parameters required for the k.p method can be defined in the `QWStrain` section of the parameter file:

```
QWStrain
{
 * Deformation Potentials (a_nu, a_c, b, C_12, C_11).
 * StrainConstant eps (formula = 1) or lattice constant
 * a0 (formula = 2) for energy shift of quantum-well
 * subbands.
 * Formula 1:
 * eps = (a_bulk - a_active)/a_bulk
 * Formula 2:
 * a0(T) = a0 + alpha (T-Tpar)
 * eps = (a_ref - a0(T))/a_ref

 * dE_c = 2 a_c (1- C12/C11) eps
 * dE_lh = 2 a_nu (1- C12/C11) eps - b (1+ 2 C12/C11) eps
 * dE_hh = 2 a_nu (1- C12/C11) eps + b (1+ 2 C12/C11) eps

        Formula = 2     # [1]

 * Mole fraction dependent model.
 * If only constant parameters are specified, those values will be
 * used for any mole fraction instead of the interpolation below.
 * Linear interpolation is used on the interval [0,1].
          * a_nu(0)  = 1                # [eV]
          * a_nu(1)  = 1.16             # [eV]
          * a_c(0)   = -5.0800e+00      # [eV]
          * a_c(1)   = -7.1700e+00      # [eV]
          * b(0)     = -1.8000e+00      # [eV]
          * b(1)     = -1.7000e+00      # [eV]
          * C_11(0)  = 8.329            # [eV]
          * C_11(1)  = 11.879           # [eV]
          * C_12(0)  = 4.526            # [eV]
          * C_12(1)  = 5.376            # [eV]
          * eps(0)   = 0.0000e+00       # [1]
          * eps(1)   = 0.0000e+00       # [1]
          * a0(0)    = 6.059e-10        # [m]
          * a0(1)    = 5.654e-10        # [m]
          * alpha(0) = 2.7400e-15       # [m/K]
          * alpha(1) = 3.8801e-15       # [m/K]
}
```

For the k.p method, `Formula = 2` must be used in the parameter file and a reference lattice constant (for example, of the substrate) must be specified by the argument `RefLatticeConst =<float>` in the `Physics-Laser-Strain` section of the command file. For non-k.p simulations, the parameter `Eps` in the parameter file is used for strain correction. It is possible to take into account the change of the lattice constant with temperature for k.p and formula 2 only.

The k.p parameters can be defined in the `BandstructureParameters` section of the parameter file:

```
BandstructureParameters
{
  * Parameters for k.p bandstructure calculation:

  * Zincblende crystals:
  * Luttinger parameters gamma_1, gamma_2, gamma_3
  * Spin-orbit split-off energy so
  * Matrix element parameters for TE and TM modes ep_te and ep_tm
```

```
     * Wurtzite crystals:
     * Effective mass parameters A1, A2, A3, A4, A5, A6
     * Spin-orbit split-off energy so
     * Crystal-field split energy cr
     * Matrix element parameters for TE and TM modes ep_te and ep_tm
     *
     *

          gamma_1 = 6.85             # [1]
          gamma_2 = 2.1              # [1]
          gamma_3 = 2.9              # [1]
          so      = 0.014            # [eV]
          ep_te   = 18.8             # [eV]
          ep_tm   = 12.4             # [eV]
          cr      = 0.019            # [eV]
          A1      = -7.2400e+00      # [1]
          A2      = -5.1000e-01      # [1]
          A3      = 6.73             # [1]
          A4      = -3.3600e+00      # [1]
          A5      = -3.3500e+00      # [1]
          A6      = -4.7200e+00      # [1]
   }
```

The k.p keywords in the command file are summarized in Table 15.156 and Table 15.157.

Table 15.156 Arguments for ManyBodyEffects section

Argument	Values		Default
`Type`	`FCT`	`SHF`	`FCT`

Table 15.157 Arguments for Bandstructure section

Argument	Values		Default
`CrystalType`	`Zincblende`	`Wurtzite`	`Zincblende`
`Order`	`nokp, kp4x4, kp6x6, kp8x8`	`nokp, kp6x6`	`nokp`
`NumKValues`	Number of values in k space (band structure is calculated up to 8e8 1/m for `Zincblende` and up to 12e8 1/m for `Wurtzite`)		`16` (`Zincblende`) `21` (`Wurtzite`)

NOTE If the wavelength search algorithm fails due to the many-body simulation or a strong strain shift of the QW subbands, the keyword `NewWavelengthSearch` can be activated in the `Math` section. This switches on a more robust algorithm that, in most cases, resolves the convergence problem.

28.9.3.1 Plotting the local band structure data

The band structures computed by the k.p method can be plotted in a similar fashion as the far field and gain plots. The syntax for this is:

```
   File{...
      Bandstructure = "bandfile"
      ...
   }
   Solve{...
      Quasistationary (
            InitialStep=0.05
            MaxStep=0.05
            Minstep=0.0003
            Plot {range=(0,1) intervals=3}
            PlotBandstructure {range=(0,1) intervals=3}
            PlotBandstructurePara {Vertex=(742 819 1150)}
            Goal {name="p_Contact" voltage=1.8})
            {
            Plugin (BreakOnFailure Iterations=8) {
               Bandstructure
               Wavelength
               Coupled{Electron Hole Poisson QWeScatter QWhScatter PhotonRate}
            }
         }
         ...
      }
```

The activation syntax is similar to that of `OptFarField` (see Section 27.9 on page 15.427) and `SaveOptField` (see Section 29.2 on page 15.472). The highlights of the syntax are:

- In the `File` section, band structure plotting is activated by the keyword `Bandstructure`, and the value assigned to it, `"bandfile"` in this case, becomes the base name for the output files of the band structures.
- In the `Solve-Quasistationary` section, the argument `range=(0,1)` in the `PlotBandstructure` keyword is mapped to the initial and final bias conditions. In this example, the initial and final (goal) `p_Contact` voltages are 0 V and 1.8 V, respectively. The number of intervals is `intervals=3`, which gives a total of four (= 3+1) modal gain plots at 0, 0.6, 1.2, and 1.8 V. In general, specifying `intervals=n` will produce (n+1) plots.
- In the `Solve-Quasistationary` section, the keyword `PlotBandstructurePara` allows users to choose the active vertex with the keyword `Vertex=(...)` from which the band structure will be plotted. These vertices have to be active vertices in the QW three-point model. For each vertex specified, the band structure $E(k)$ is plotted for the different subbands as well as the wavefunctions $\Phi(x)$ on the nonlocal line crossing the vertex. The entries in `Vertex=(...)` are DESSIS vertex indices and they can be obtained from Tecplot-ISE. Set the environment variable `TEC_GRID_DEBUGGING`, run Tecplot-ISE, and select **Plot** > **Label Points** > **Cells** > **Show Cell Labels** > **Show Variable Value** > **VertexIndex**.

  The units in the plot files must be [1/m] and [eV] for the $E(k)$ plot, and [m] and [$1/\sqrt{m}$] for the $\Phi(x)$ plot.
- The output files in this example are `bandfile_000000_des.plt` ... `bandfile_000003_des.plt`. These files can be viewed with INSPECT.

28.10 Importing external gain with PMI

DESSIS can import external stimulated gain data through the physical model interface (PMI). The gain PMI concept is illustrated in Figure 15.111 on page 15.467.

DESSIS calls the user-written gain calculations through the PMI with the variables: electron density n, hole density p, electron temperature eT, hole temperature hT, and transition energy E. The user-written gain calculation then returns the gain g and the derivatives of gain with respect to n, p, eT, and hT to DESSIS. The derivatives are required to ensure proper convergence of the Newton iterations. In this way, the user-import gain is made self-consistent within the laser simulation.

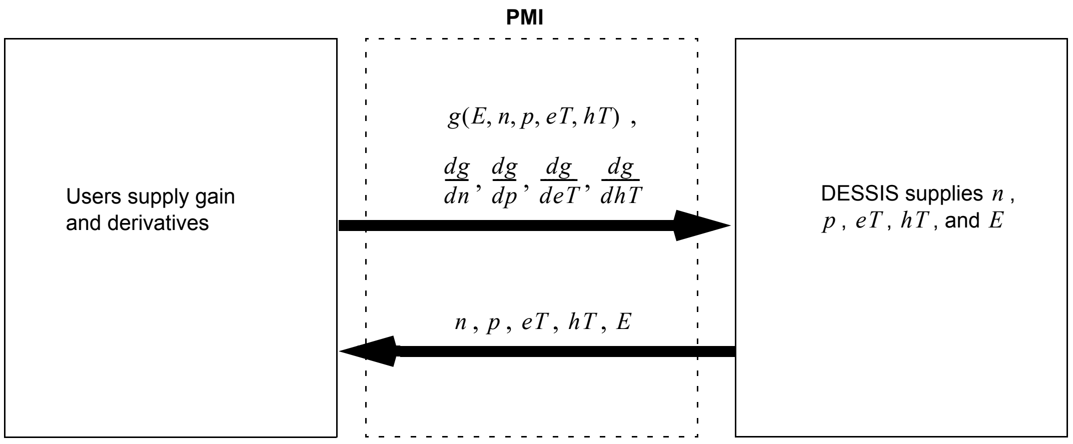

PMI
$g(E, n, p, eT, hT)$,
$\frac{dg}{dn}, \frac{dg}{dp}, \frac{dg}{deT}, \frac{dg}{dhT}$
Users supply gain and derivatives
DESSIS supplies n , p , eT , hT , and E
n , p , eT , hT , E

Figure 15.111 Concept of the gain PMI

28.10.1 Implementation of the gain PMI

The PMI uses the object-orientation capability of the C++ language (see Chapter 33 on page 15.535). A brief outline is given here of the gain PMI.

In the DESSIS header file `PMIModels.h`, the following base class is defined for gain:

```
class PMI_EXTERNAL PMI_StimEmissionCoeff : public PMI_Dessis_Interface {
public:
 PMI_StimEmissionCoeff (const PMI_Environment& env);
 virtual ~PMI_StimEmissionCoeff ();

 virtual void Compute_rstim
 (double E,
   double n,
   double p,
   double et,
   double ht,
   double& rstim) = 0;

 virtual void Compute_drstimdn
 (double E,
   double n,
   double p,
   double et,
   double ht,
   double& drstimdn) = 0;

 virtual void Compute_drstimdp
 (double E,
   double n,
   double p,
   double et,
   double ht,
   double& drstimdp) = 0;

 virtual void Compute_drstimdet
 (double E,
   double n,
   double p,
   double et,
   double ht,
   double& drstimdet) = 0;

 virtual void Compute_drstimdht
 (double E,
   double n,
   double p,
   double et,
   double ht,
   double& drstimdht) = 0;
};
```

To implement the PMI model for gain, the user must declare a derived class in the user-written header file:

```
#include "PMIModels.h"

class StimEmissionCoeff : public PMI_StimEmissionCoeff {
 // User-defined variables for his/her own routines
 private:
    double a, b, c, d;

 public:
    // Need a constructor and destructor for this class
    StimEmissionCoeff (const PMI_Environment& env);
    ~StimEmissionCoeff ();

    // --- User needs to write the following routines in the .C file ---
    // The value of the function is return as the last pointer argument

    // stimulated emission coeff value
    void Compute_rstim (double E,
                        double n,
                        double p,
                        double et,
                        double ht,
                        double& rstim);

    // derivative wrt n
    void Compute_drstimdn (double E,
                           double n,
                           double p,
                           double et,
                           double ht,
                           double& drstimdn);

    // derivative wrt p
    void Compute_drstimdp (double E,
                           double n,
                           double p,
                           double et,
                           double ht,
                           double& drstimdp);

    // derivative wrt eT
    void Compute_drstimdet (double E,
                            double n,
                            double p,
                            double et,
                            double ht,
                            double& drstimdet);
    // derivative wrt hT
    void Compute_drstimdht (double E,
                            double n,
                            double p,
                            double et,
                            double ht,
                            double& drstimdht);
};
```

Next, the user must write the functions `Compute_rstim`, `Compute_drstimdn`, `Compute_drstimdp`, `Compute_drstimdet`, and `Compute_drstimdht` to return the values of the stimulated emission coefficient and its derivatives to DESSIS using this gain PMI. If users have, for example, a table of gain values, they must implement the above functions to interpolate the values of the gain and derivatives from the table.

The spontaneous emission coefficient can also be imported using the PMI. The implementation is exactly the same as the stimulated emission coefficient, and users only need to replace `StimEmissionCoeff` with `SponEmissionCoeff` in the above code.

CHAPTER 29 Additional features of laser or LED simulation

A host of features is available to aid the simulation of lasers and LEDs.

29.1 Free carrier loss

The free carrier loss is caused by plasma-induced effects and contributes to the total optical loss as shown in (Eq. 15.478). The free carrier loss can be modeled by:

$$L_{carr} = \iint(\alpha_n n + \alpha_p p)|\Psi|^2 dV \tag{15.612}$$

where the loss is linearly dependent on the electron and hole carrier densities. This model is activated by the keyword `FreeCarr` in the `Physics-Laser` section of the command file:

```
Plot {...
   efreecarr
   hfreecarr
}
...
Physics {...
   Laser (...
      Optics (...
         FEVectorial (...
         )
      )
      FreeCarr
   )
}
```

In this example, `eFreeCarr` and `hFreeCarr` in the `Plot` section enable the electron and hole contributions to the free carrier loss to be plotted. The coefficients for the free carrier loss, α_n and α_p, for each material region are specified in the parameter file:

```
FreeCarrierAbsorption
{
 * Coefficients for free carrier absorption:
 * alpha_n for electrons,
 * alpha_p for holes

 * FCA = (alpha_n * n + alpha_p * p) * Light Intensity
 * Mole fraction dependent model.
 * If only constant parameters are specified, those values will be
 * used for any mole fraction instead of the interpolation below.
 * Linear interpolation is used on the interval [0,1].
        fcaalpha_n(0)    = 4.0000e-18    # [cm^2]
        fcaalpha_n(1)    = 5.0000e-18    # [cm^2]
        fcaalpha_p(0)    = 8.0000e-18    # [cm^2]
        fcaalpha_p(1)    = 9.0000e-18    # [cm^2]
}
```

29.2 Saving and loading optical modes

The calculation of the optical modes of a laser can be time-consuming especially for large geometries. To save simulation time, transverse optical mode patterns can be saved and loaded from data files. However, when the optical modes are loaded from a file, the optical modes are assumed to be constant throughout the simulation. As a result, self-consistent iteration between the optics and electronics is not possible in this case.

29.2.1 Saving optical modes on optical or electrical mesh

The optical field can be saved on the grid of the optical device (referred to as the optical mesh) in a dual-grid simulation by using the keyword `SaveOptField`. To understand more about dual-grid simulation (see Section 25.2.2 on page 15.378). If the optical mesh is not found, the keyword `SaveOptField` saves the optical field to the only grid that is available – the electrical mesh as it is in Section 25.2.1 on page 15.373. The activation keyword is `SaveOptField` in the `File` and `Solve-quasistationary` sections of the command file:

```
File {...
      SaveOptField = "laserfield"
}
...
Solve {...
   quasistationary (
        SaveOptField { range=(0,1) intervals=3 }
        Goal({name="p_Contact" voltage=1.8})
        {...}
}
```

Refer to Section 25.2.1 to see where this syntax is inserted in the command file. The `SaveOptField` has the same format as the `OptFarField` (see Section 27.9 on page 15.427) and `GainPlot` (see Section 29.4 on page 15.475). In this way, users can track the evolution of the optical field as the bias increases. With regard to the syntax and its output:

- In the `File` section, saving the optical field is activated by the keyword `SaveOptField`. The value assigned to it, `"laserfield"` in this case, becomes the base name for the output files of the optical fields.
- In the `Solve-quasistationary` section, the argument `range=(0,1)` in the `SaveOptField` keyword is mapped to the initial and final bias conditions. In this example, the initial and final (goal) `p_Contact` voltages are 0 V and 1.8 V, respectively. The number of `intervals=3` , which gives a total of four (= 3+1) optical field saved at 0 V, 0.6 V, 1.2 V, and 1.8 V. In general, specifying `intervals=n` will produce (n+1) sets of optical field files.
- The output files in this example are `laserfield_000000_mode0_int.dat`, `laserfield_000000_mode0_real.dat`, and `lasefield_000000_mode0_imag.dat`. The first is the optical field intensity, and the other two are the real and imaginary parts of the vectorial optical fields. If the scalar optical solver `FEScalar` is used, the optical field is saved in the x-component of the real and imaginary files.
- For an LED simulation, only the intensity file is produced because the LED raytracing contains no vectorial or phase information.

29.2.2 Loading optical modes from arbitrary mesh

The optical mode that is read into DESSIS with the keyword `OptField<n>` must strictly be on the same DF–ISE mesh as the electrical device. If the optical field has been saved on another mesh (for example, in an optics stand-alone simulation on a different optical mesh), the field must be interpolated onto that of the electrical mesh before DESSIS can load it. This is accomplished by using the ISE tool DIP. For example, to interpolate the 2D optical intensity for an edge-emitting laser, the following commands for `dipsh` will perform the interpolation:

```
set A [dip_mesh2D -args NULL optical_grid.grd mode_0_int.dat]
set B [dip_mesh2D -args NULL electronic_grid.grd 0]
$B importDatasets $A new OpticalIntensity
$B writeDatasets mode_0_interpolated_int.dat 0
```

These commands can be saved into a file, for example, `dipcommands.cmd`, and the DIP interpolation routine can be called by `dipsh dipcommands.cmd`. To interpolate the real and imaginary parts of the vectorial optical fields, the corresponding `_real.dat` or `_imag.dat` files can be used instead, and `OpticalIntensity` is replaced with `OpticalField` in the above `dipsh` commands.

NOTE As stated in Section 29.2 on page 15.472, when optical fields are saved, three files containing the intensity (`_int.dat`), and the real (`_real.dat`) and imaginary (`_imag.dat`) vectorial optical fields are produced.

Users can selectively load up to ten possible vectorial optical fields, which have been computed separately. The activating keyword is `OptField<n>` and the following example shows the syntaxes for loading scalar and vectorial optical fields.

Loading scalar optical fields

```
File {...
   OptField0 = "mode_0"
   OptField1 = "mode_1"
}
...
Physics {...
   Laser (...
      Optics (...
         FEScalar(
                Polarization = (TE TM)
                ModeNumber=2
         )
      )
   )
}
```

Loading vectorial optical fields

```
File {...
   OptField0 = "mode_0"
   OptField1 = "mode_1"
   OptField2 = "mode_2"
}
...
Physics {...
   Laser (...
      Optics (...
         FEVectorial( ModeNumber=3 )
      )
   )
}
```

Some comments about these syntaxes are:

- For example, if `"mode_0"` is specified by the keyword `OptField0` in the `File` section, DESSIS searches for `mode_0_int.dat`, `mode_0_real.dat`, and `mode_0_imag.dat` as the input files to load the intensity as well as the real and imaginary parts of the vectorial optical field. In the case of scalar fields, the scalar optical field is stored as the x-component of the vectorial field files.
- The optical solver type in the `Physics-Laser-Optics` section must be specified as `FEScalar` or `FEVectorial` so that DESSIS knows which type of optical field to read.
- The number of optical field files specified must match the number of modes specified by `Modenumber`.
- For scalar optical fields, the type of `Polarization` must be set for each mode that is read. For vectorial fields, the optical polarization angles are automatically computed.
- After the optical fields are read, DESSIS normalizes them for use with the photon rate equation.

NOTE When optical modes are read from files, self-consistent simulation between the optics and electronics is not possible.

29.2.3 Obsolete optical intensity save and load options

The keywords for loading and saving optical intensities, `SaveOptPattern` and `OptPattern`, are obsolete and will be discontinued in Release 10.0.

29.3 Symmetry considerations

Imposing symmetry of the device in laser simulations requires the treatment of symmetry in both the optical and electrical parts of the problem. In particular, symmetry changes the boundary conditions for the optical solvers and must be handled carefully. See Chapter 27 on page 15.399 for a detailed discussion about symmetry in the optics.

Conversely, symmetry in the electrical part of the problem is simpler. With the finite box method used in the simulation, the Neumann boundary condition is implicitly imposed at all boundaries that are not defined as the contacts. Therefore, there is no need for the user to set the boundary specifically for symmetric structures. However, an additional area factor of 2 must be specified in the `Physics` section:

```
Physics {...
   AreaFactor = 2
   Laser (...
      Optics(
         FEVectorial(...
                     Symmetry = Symmetric
         )
      )
   )
}
```

This adjusts the total electrode area and the optical output power, both of which double in the symmetric simulation mode.

29.3.1 Cylindrical symmetry

VCSEL structures are generally assumed to be cylindrically symmetric, and cylindrical symmetry in DESSIS is managed in a slightly different way. The 2D plane whereby the device is drawn is treated as the (ρ, z) plane in cylindrical symmetry.

In addition to the specification of `Coordinates=Cylindrical` in the `Physics-Laser-Optics-FEVectorial` section, the keyword `Cylindrical` must also be added to the `Math` section:

```
Physics {...
   AreaFactor = 1
   Laser (...
      Optics(
         FEVectorial(...
                     Coordinates = Cylindrical
         )
      )
      VCSEL()          # specify this is a VCSEL simulation
   )
}
...
Math {...
   Cylindrical
}
```

This is to ensure that the area and volume computations associated with each vertex is based on cylindrical symmetry.

NOTE In the cylindrical symmetry case, it is not necessary to specify an `AreaFactor` of 2.

29.4 Plotting gain

Modal or material gain is an important parameter in laser operations. It affects the threshold current, modulation response, external efficiency, lasing wavelength, and so on. In the simulation, the modal or material gain can be monitored in three ways: as a function of bias, spatial distribution, and a function of energy. Refer to Section 25.2.1 on page 15.373 while proceeding through these gain-plotting options.

29.4.1 Modal gain as a function of bias

The modal gain for the lasing frequency is automatically output in the `Current` file in a laser simulation if the current file has been specified:

```
File {...
   Current = "multiqw_curr"
}
```

At the end of the simulation, the file `multiqw_curr_des.plt` is produced. With INSPECT, users can plot `OpticalGain` as a function of bias current, voltage, and so on. This modal gain [1/cm] is the total gain of the device at the lasing frequency. In multimode simulations, a modal gain curve corresponds to each mode, and the modal gain is taken at the respective lasing frequencies of the modes.

29.4.2 Material gain in the active region

The peak value of the local material gain at each vertex of the active region can be extracted by including the keyword `MatGain` in the `Plot` section:

```
File {...
   Grid = "mesh_mdr.grd"
   Plot = "multiqw_plot"
}
...
Plot {...
   MatGain
}
...
```

At the end of the simulation, the file `multiqw_plot_des.dat` will be produced. This file can be used in Tecplot-ISE in conjunction with the grid file, `mesh_mdr.grd`, to view the peak material gain at each spatial location of the active region.

29.4.3 Modal gain as a function of energy/wavelength

One important way to look at the gain is to plot the modal gain as a function of the energy. This is possible by including gain-plotting keywords in the `File` and `Solve-quasistationary` sections of the command file:

```
File {...
   ModeGain = "gain"
}
...
Solve {...
   quasistationary (...
        # ----- Specify plot gain parameters -----
        PlotGain { range=(0,1) intervals=3}
        PlotGainPara{range=(1.22,1.32) intervals=120}# energy range [eV], discretization
        Goal {name="p_Contact" voltage=1.8})
        {...}
}
```

The activation syntax is similar to that of `OptFarField` (see Section 27.9 on page 15.427) and `SaveOptField` (see Section 29.2 on page 15.472). The highlights of the syntax are:

- In the `File` section, modal gain-plotting is activated by the keyword `ModeGain`, and the value assigned to it, `"gain"` in this case, becomes the base name for the output files of the modal gain as a function of energy.
- In the `Solve-quasistationary` section, the argument `range=(0,1)` in the `PlotGain` keyword is mapped to the initial and final bias conditions. In this example, the initial and final (goal) `p_Contact` voltages are 0 V and 1.8 V, respectively. The number of `intervals=3` , which gives a total of four (= 3+1) modal gain plots at 0 V, 0.6 V, 1.2 V, and 1.8 V. In general, specifying `intervals=n` will produce (n+1) plots.
- In the `Solve-quasistationary` section, the keyword `PlotGainPara` allows users to choose the energy range of the gain curve to plot. In this example, the range has been chosen as 1.22–1.32 eV, and 120 discretization points are to be used.
- The output files in this example are `gain_gain_000000_des.plt`, ..., `gain_gain_000003_des.plt`. These files contain the modal gains as a function of energy or wavelength, and can be viewed with INSPECT.

29.5 Refractive index, dispersion, and optical loss

The refractive index (dielectric constant) is a function of temperature, carrier density, and wavelength. The temperature dependence is assumed to be linear and the change in refractive index due to carriers is attributed to the free carrier plasma effect [155].

The user can specify the refractive index dependence on temperature and carrier density with the keywords `TemperatureDep` and `CarrierDep` in the `Physics-Laser` section of the command file:

```
Physics {...
   Laser (...
      RefractiveIndex(TemperatureDep CarrierDep)
   )
}
```

The user can select either or both type of dependence. If none is chosen (that is, there are no keywords), DESSIS assumes that the refractive index is a constant value.

NOTE The refractive index and its associated temperature parameters for each material region can be changed by the user in the parameter file.

29.5.1 Temperature dependence of refractive index

The temperature dependence of the refractive index follows the relation:

$$n(T) = n \cdot (1 + \alpha(T - T_{par})) \tag{15.613}$$

and the coefficients can be changed in the parameter file:

```
RefractiveIndex
{ * Optical Refractive Index
  * refractiveindex() = refractiveindex * (1 + alpha * (T-Tpar))
        Tpar    = 3.0000e+02            # [K]
        refractiveindex = 3.60e+00
        alpha   = 0.0000e-04            # [1/K]
}
```

29.5.2 Carrier density dependence of refractive index

The change in refractive index due to free carriers [155] is:

$$\Delta n = -CarrDepCoeff \times \frac{e^2\lambda^2}{8\pi^2 c^2 \varepsilon_0 n_g}\left(\frac{n}{m_e} + \frac{p}{m_h}\right) \tag{15.614}$$

where `CarrDepCoeff` is introduced as a tuning parameter and has a default value of 1. λ is the lasing wavelength and n_g is the refractive index. n_g can also change if it is specified as `TemperatureDep`.

In the case of QW carriers, the heavy-hole mass is modified to become [155]:

$$m_h = \frac{m_{hh}^{1.5} + m_{lh}^{1.5}}{\sqrt{m_{hh}} + \sqrt{m_{lh}}} \tag{15.615}$$

The activating syntax in the command file is:

```
Physics {...
   Laser (...
      Optics (...)
         RefractiveIndex(CarrierDep)                            # use lasing wavelength of mode0
#        RefractiveIndex(CarrierDep(LasingWavelength=777.77))    # fixed wavelength [nm]
   )
}
```

There is an option to fix a `LasingWavelength` if required. Otherwise, DESSIS automatically uses the lasing wavelength that it computes in the simulation. The tuning parameter `CarrDepCoeff` is defined for each region and can be input in the parameter file:

```
RefractiveIndex
{ *  Optical Refractive Index
        refractiveindex = 3.893 # [1]
  * Mole fraction dependent model.
  * If just above parameters are specified, then its values will be
  * used for any mole fraction instead of an interpolation below.
  * The linear interpolation is used on interval [0,1].
        refractiveindex(1) = 3.51     # [1]

  * Tune the region-wise carrier dependence (plasma effect)
  *                            e^2.lambda^2         (  n     p  )
  * del_n = - CarrDepCoeff * ------------------------- ( --- + --- )
  *                       8pi^2.c^2.epsilon0.n_refr ( m_e   m_h )
  * Default of CarrDepCoeff is 1.

      CarrDepCoeff = 1.0      # [1]
      CarrDepCoeff(0) = 0.5
      CarrDepCoeff(1) = 1.0
}
```

If `CarrDepCoeff` is not entered in the parameter file, it is assumed to be the default value of 1 during the simulation. `CarrDepCoeff` is a mole fraction–dependent parameter, so it works the same way as the other mole fraction–dependent parameters.

29.5.3 Wavelength dependence and absorption of refractive index

Wavelength-dependent optical absorption can also be included by using the `ODB` (Optik database) feature in the optical solver section of the command file:

```
Physics {...
   Laser (...
      Optics (...
         FEVectorial (...
                   Absorption(ODB)
                )
             )
          )
       }
```

This feature is also available for the other optical solvers such as `FEScalar`, `TMM1D`, and `EffectiveIndex`. In this case, the user must create a corresponding `ODB` table for each material region in the parameter file, for example:

```
TableODB
{ * Table format of the Optik DataBase
  * complex refractive index n + i*k (unitless)
  * refractive index = n, absorption coefficient = 4*pi*k/wavelength
  * WAVELEN(um) n k
  0.5904 3.940 0.240
  0.6199 3.878 0.211
  0.6526 3.826 0.179
  0.6888 3.785 0.151
  0.7293 3.742 0.112
  0.7749 3.700 0.091
  0.8266 3.666 0.080
  0.8856 3.614 0.0017
  0.9184 3.569 0.0
  1.0332 3.492 0.0
  1.1271 3.455 0.0
  1.2399 3.423 0.0
  1.3776 3.397 0.0
  1.5498 3.374 0.0
  1.7712 3.354 0.0
  2.0664 3.338 0.0
  2.4797 3.324 0.0
}
```

29.6 Transient simulation

Transient simulation is important in the tracking of the time evolution of carrier dynamics and photon output fluctuations in a laser diode. DESSIS has an option to perform the transient simulation of edge-emitting lasers, VCSELs, and LEDs. A small step bias is applied at the input and, through Newton iterations, the time-varying continuity equations and photon rate equations are solved self-consistently with the Poisson equation, QW scattering equations, and Schrödinger equation.

Upon the application of the step bias, the time steps are increased in small intervals to trace the dynamics of the carriers and photons in the transient simulation. At each time step, the full set of variables everywhere in the device can be saved and plotted. In this way, users can calculate which feature of the device is hampering the faster modulation of the device.

Transient simulations performed at different biases yield different dynamics, so users must perform a quasistationary simulation first to reach the required bias current or voltage level, after which the transient simulation is activated. The syntax for transient simulation of general devices is described in Section 2.9.4 on page 15.62. Here, the emphasis is on explaining the syntax that is related to Section 25.2.1 on page 15.373.

29.6.1 Syntax for laser transient simulation

The transient simulation can be activated by the keyword `Transient` in the `Solve` section of the command file:

```
Solve {
   # ----- Get initial guesses, coupled means Newton's iteration -----
   Poisson
   coupled {Hole Electron Poisson }
   coupled {Hole Electron QWhScatter QWeScatter Poisson }
   coupled {Hole Electron QWhScatter QWeScatter Poisson PhotonRate}

   # ----- Ramping the voltage to 1.8 V -----
   quasistationary (
        # ----- Specify ratio step size of voltage ramp -----
        InitialStep = 0.001
        MaxStep = 0.05
        Minstep = 1e-5

        # ----- Specify the final voltage ramp goal -----
        Goal {name="p_Contact" voltage=1.8})
        {
           # ----- Gummel iterations for self-consistency of Optics -----
           Plugin(BreakOnFailure){
                # --- Newton iterations for coupled equations -----
                Coupled { Electron Hole Poisson QWeScatter QWhScatter
                          PhotonRate }
                Wavelength
                Optics
           }
        }

   # ----- At 1.8V, perform transient simulation -----
   transient (
              # ----- Specify the starting and ending time -----
              InitialTime = 0.0         # [s]
              FinalTime = 2.0e-9        # [s]

              # ----- Control the time step size -----
              InitialStep = 1.0e-3
              MinStep = 1.0e-3
              MaxStep = 1.0e-1

              # ----- Save the plot variables at specified intervals -----
              Plot { range=(0,1) intervals=4 }
        )
        {
           Plugin(BreakOnFailure){
                Coupled { Electron Hole Poisson QWeScatter QWhScatter
                          PhotonRate }
#               Wavelength
#               Optics
           }
        }
}
```

The above syntax has been extracted from Section 25.2.1 on page 15.373:

- A quasistationary simulation is performed first to ramp up the operating voltage to a bias of 1.8 V before the transient simulation is activated.
- In the `transient` section, the user must specify the start and end time. The response of the carriers and photons subjected to a small step bias input will generally settle to a steady state in the time frame of a few thousand picoseconds. This settling time depends on bias conditions and the type of laser diode. Users are encouraged to use different values of `FinalTime` so that the important parts of the transient response are observed.
- In the `transient` section, the user must specify the time-step size as well in the same format as is used to specify the bias step size in the quasistationary simulation. In this case, the time-step size is the fraction indicated in `InitialStep`, `MinStep`, and `MaxStep` multiplied by (`FinalTime - InitialTime`), which gives 2, 2, and 200 picoseconds, respectively. The scattering time for the carriers is in the order of 1×10^{-13} s, so a time step shorter than this time will not be meaningful. Of course, if the time step is too big, it cannot resolve the finer features of the transient response.
- In the `transient` section, the `Plot` statement enables users to save the variables that have been defined in the `Plot` section at required intervals of the transient simulation. In this example, `intervals=4` produces five plot files at the time intervals of 0, 400, 800, 1200, 1600, and 2000 picoseconds.
- The transient response computed is saved in the current file. The major results of laser output are saved as a function of time in this current file. The user can then perform a FFT of the time response to obtain the modulation response of the laser diode.

29.7 Performing a temperature simulation

There are a few different types of temperatures, but DESSIS only treats two types:

- Lattice temperature, which describes the vibrational states of the crystal lattice
- Carrier temperatures, which determine the distribution of the excited carriers

The lattice temperature is solved by the lattice temperature model described in Section 4.2.3 on page 15.128. The carrier temperatures are solved by the hydrodynamic equations discussed in Section 4.2.4 on page 15.130.

NOTE For stability reasons, the QW scattering model (see Section 28.2.3 on page 15.441) must be activated when the lattice temperature or hydrodynamic models are solved in a laser simulation.

29.7.1 Lattice temperature simulation

To solve the lattice temperature model, the following three steps must be taken:

1. A `thermode` must be defined in the grid file and command file.
2. In the `Physics` section, the keyword `Thermodynamic` is optional. If it is included, a lattice temperature gradient term is appended to the carrier current flux density.
3. In the `Solve` section, the keyword `LatticeTemperature` or `Temperature` must be added to the `Coupled` statement.

These steps are embedded in the following syntax:

```
# ----- Need to specify where to impose Dirichlet boundary condition for temperature solution -----
Thermode {
    { Name="top_thermode" AreaFactor=200 Temperature=300 SurfaceResistance = 0.14}
    { Name="bot_thermode" AreaFactor=200 Temperature=300 SurfaceResistance = 0.09}
}

Plot {
   # ----- Temperature variables -----
   LatticeTemperature
}
...
Physics {...
   Laser (...
      Optics (...)
   )
   # ------ Turn on thermodynamic simulation ------
       Thermodynamic
       RecGenHeat
}
...
Solve {
   # ----- Get initial guesses, coupled means Newton's iteration -----
   Poisson
   coupled {Hole Electron QWhScatter QWeScatter Poisson }
   coupled {Hole Electron QWhScatter QWeScatter Poisson PhotonRate}
   coupled {Hole Electron QWhScatter QWeScatter Poisson PhotonRate
            LatticeTemperature}

   # ----- Ramping the voltage -----
   quasistationary (
        # ----- Specify ratio step size of voltage ramp -----
        InitialStep = 0.001
        MaxStep = 0.05
        Minstep = 1e-5

        # ----- Specify the final voltage ramp goal -----
        Goal {name="p_Contact" voltage=1.8})
        {
           # ----- Gummel iterations for self-consistency of Optics -----
           Plugin(BreakOnFailure){
                # --- Newton iterations for coupled equations -----
                Coupled { Electron Hole Poisson QWeScatter QWhScatter
                          PhotonRate LatticeTemperature}
                Wavelength
                Optics
           }
        }
}
```

Refer to Section 25.2.1 on page 15.373 for the rest of the laser simulation syntaxes, which have been omitted here. Some comments about the above syntax are:

- A `thermode` is a boundary where the Dirichlet boundary condition is set for the lattice temperature equation. At all other boundaries without a thermode, Neumann boundary condition is assumed. It can be set in the same way as the contact electrodes are set in MDRAW or DEVISE, and this is shown in Figure 15.112 on page 15.483. The `SurfaceResistance` [cm^2 K/W] can be used to tune the lattice temperature profile in the device.

- In the `Physics` section, the keyword `Thermodynamic` contributes an additional term (gradient of the lattice temperature) to the electron and hole flux densities. The keyword `RecGenHeat` adds heat produced or absorbed by generation–recombination to the lattice temperature equation.
- In the `Solve` section, the keyword `LatticeTemperature` in the `Coupled` statement adds the lattice temperature equation to the Newton system to be solved iteratively.

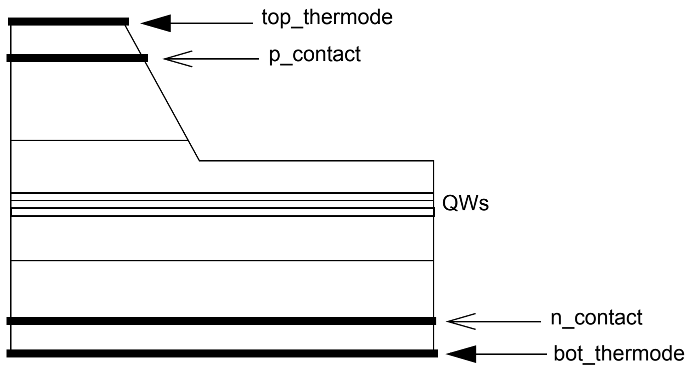


Figure 15.112 Setting thermodes in the same way as setting contacts; labeled top_thermode and bot_thermode, respectively

29.7.2 Carrier temperature simulation

The carrier temperatures can be solved by the hydrodynamic model, and the activation syntaxes are located in similar locations as the lattice temperature model:

```
Plot {
   # ----- Temperature variables -----
   eTemperature
   hTemperature
}
...
Physics {...
   Laser (...
      Optics (...)
   )
   # ----- Specify ambient device temperature -----
   Temperature = 298
   # ------ Turn on hydrodynamic simulation ------
   Hydrodynamic
   RecGenHeat
}
...
Solve {
   # ----- Get initial guesses, coupled means Newton’s iteration -----
   Poisson
   coupled {Hole Electron QWhScatter QWeScatter Poisson }
   coupled {Hole Electron QWhScatter QWeScatter Poisson PhotonRate}
   coupled {Hole Electron QWhScatter QWeScatter Poisson PhotonRate
              eTemperature hTemperature}

   # ----- Ramping the voltage -----
   quasistationary (
        # ----- Specify ratio step size of voltage ramp -----
        InitialStep = 0.001
        MaxStep = 0.05
        Minstep = 1e-5
```

```
          # ----- Specify the final voltage ramp goal -----
          Goal {name="p_Contact" voltage=1.8}}
          {
             # ----- Gummel iterations for self-consistency of Optics -----
             Plugin(BreakOnFailure){
                  # --- Newton iterations for coupled equations -----
                  Coupled { Electron Hole Poisson QWeScatter QWhScatter
                            PhotonRate eTemperature hTemperature}
                  Wavelength
                  Optics
             }
          }
  }
```

Some comments about the above syntax are:

- The user does not need to specify any thermode if the hydrodynamic model is solved without the lattice temperature equation. However, it is possible to solve both the lattice temperature and hydrodynamic models together. The user must include the keywords for both models.
- In the `Plot` section, users can choose to save the electron and hole temperatures everywhere in the device with the keywords `eTemperature` and `hTemperature`.
- In the `Physics` section, if the lattice temperature is not solved, the user can specify an ambient device lattice temperature with the keyword `Temperature=<float>`. The hydrodynamic model is activated by the keyword `Hydrodynamic` in this section. The keyword `RecGenHeat` adds the heat produced or absorbed as a result of carrier generation–recombination to the hydrodynamic equations.
- In the `Solve` section, the hydrodynamic model requires two keywords, `eTemperature` and `hTemperature`, to be included in the `Coupled` statement.

29.8 Optics stand-alone option

In many cases, the initial phase of design of a laser diode involves the optimization of the geometry of the laser structure to achieve specific optical mode properties, for example, mode shape. DESSIS provides an option to run the optical solver without invoking the entire laser simulation.

An example of a stand-alone optical simulation of an edge-emitting laser, waveguiding structure is:

```
# ----- Optics Stand-alone command file -----

# ----- Specify only a dummy electrode -----
Electrode {
   { Name="dummy"   voltage=0.0 }
}

# ----- Tell Dessis where to read/save the parameters/results -----
File {
   Grid = "optmesh_mdr.grd"
   Doping = "optmesh_mdr.dat"
   Parameter = "des_las.par"
   Plot = "opt_plot"
   Output = "log"
   # ----- Compute and save the farfield -----
   OptFarField = "farfield"
   # ----- Save the optical field -----
   SaveOptField = "optmode"
}

Plot {
   refractiveindex
}

# ----- Specify the material region properties, as usual -----
Physics (region="pbulk") { MoleFraction(xfraction=0.28) }
Physics (region="nbulk") { MoleFraction(xfraction=0.28) }
Physics (region="psch") { MoleFraction(xfraction=0.09) }
Physics (region="nsch") { MoleFraction(xfraction=0.09) }
Physics (region="barr") { MoleFraction(xfraction=0.09) }
# ----- Quantum Wells -----
Physics (region="qw1") {
   MoleFraction(xfraction = 0.8 Grading=0.00)
   Active       # keyword to specify active region
}
Physics (region="qw2") {
   MoleFraction(xfraction = 0.8 Grading=0.00)
   Active
}

# ----- Major difference from the laser command file -----
Physics {
    Optics (
        FEVectorial ( EquationType = Waveguide           # or Cavity
                      Symmetry = Nonsymmetric            # or Symmetric or Periodic
                      LasingWaveLength = 656             # [nm]
                      TargetEffectiveIndex = (3.45 3.42 3.34)
                      Boundary = ("Type2" "Type1" "Type2")
                      ModeNumber = 3
        )
    )
    HeteroInterfaces
}

Solve { Optics }
```

Compare this code to Section 25.2.1 on page 15.373. The most significant difference in the optics stand-alone simulation is that the entire `Laser` section has been removed. Other notable changes are:

- In the `Electrode` statement, a dummy electrode must be specified in order to conform to the DESSIS input format. It has no other purpose in the optical simulation.
- In the `File` section, the user can choose to plot the far field with the keyword `OptFarField`. For the optics stand-alone case, the far-field parameters are set internally. The range of observation angles for the far field is $[-90°, 90°]$ in both directions. If a scalar optical solver (for example, `FEScalar`) is chosen, only the scalar far field is produced.
- In the `File` section, the user must specify the base name for the optical-field files with the keyword `SaveOptField`. These files can be read into the laser simulation (see Section 29.2 on page 15.472).
- The optical-active region is specified in the material region `Physics` statement by the keyword `Active`. This is important so that the optical confinement factor can be computed within DESSIS.
- In the `Optics` section, the optical mode solver type (`FEScalar`, `FEVectorial`, `TMM1D`, or `EffectiveIndex`) is specified. The arguments for these mode solver types are discussed in Section 27.3 on page 15.401, Section 27.6 on page 15.411, and Section 27.7 on page 15.413.
- In the `Solve` section, the stand-alone calculation of the optical mode solver is activated by the sole keyword `Optics`.

29.9 Switching from voltage to current ramping

Before the lasing threshold, the voltage varies almost linearly with the spontaneous output power. At and beyond the lasing threshold, a small increase in voltage will lead to an exponential-order increase in the laser output power. Therefore, it is desirable to switch between voltage and current bias ramping in a laser simulation. This is achieved using the following syntax in the `Solve` section of the command file:

```
Solve {
   # --- Solve for initial guesses of the coupled system ---
   Poisson
   Coupled {Poisson Hole Electron}
   Coupled {Poisson Hole Electron QWeScatter QWhScatter}
   Coupled {Poisson Hole Electron QWeScatter QWhScatter PhotonRate}

   # --- Ramp using voltage bias to near lasing threshold ---
   Quasistationary (
        InitialStep = 0.01
        MaxStep = 0.1
        Minstep = 1e-9
        Goal {name="p_contact" voltage=1.3})
        {
           Plugin(BreakOnFailure){
                Coupled { Poisson Hole Electron QWeScatter QWhScatter
                          PhotonRate }
                Optics
                Wavelength
           }
        }

   # --- Change the p_contact from voltage to current type ---
   Set ("p_contact" mode current)

   # --- Then ramp using current bias ---
   Quasistationary (
        InitialStep = 0.01
        MaxStep = 0.05
        Minstep = 1e-8
        Goal {name="p_contact" current=2.5e-4})
        {
           Plugin(BreakOnFailure){
                Coupled { Poisson Hole Electron QWeScatter QWhScatter
                          PhotonRate }
                Optics
                Wavelength
           }
        }
}
```

The threshold voltage of a laser diode can be estimated approximately by considering the laser diode as a p-i-n diode with the quantum well as a strong recombination center. In order for diffusion current to flow into the quantum well, the intrinsic Fermi levels of the bulk region near the p or n contact must approximately align with that of the quantum well. This means a voltage that is approximately the difference in these Fermi levels must be applied for the device to start the current flow towards the quantum wells to fill them.

Of course, the lasing will not start until the quantum wells are filled to a level such that population inversion is achieved and the threshold losses are overcome. Resistance of the semiconductor layers will also contribute to the voltage. Nevertheless, this consideration will give a rough estimate of the threshold voltage for current flow in the laser diode.

29.10 Scripts

There is a list of scripts that are written in Tcl and can be used to extract various useful parameters from laser and LED simulations, as well as to control different tools in the ISE software suite. The laser-related or LED-related scripts available from ISE Technical Support are:

- Automatic extraction of the threshold current
- Calculation of the slope efficiency of the L–I curve
- Automatic addition of DBR layers in the VCSEL structure
- Automatic addition of PML to the laser structure
- Generation of multiple–quantum well (MQW) edge-emitting lasers
- Generation of MQW VCSEL structures
- Ternary-material and quaternary-material parameter calculations
- Extraction of far-field angle from far-field patterns

CHAPTER 30 Simulation of different laser types and LEDs

30.1 Overview

DESSIS can simulate different laser diode types and LEDs with different geometry. In this section, different types of edge-emitting laser, VCSEL, and LED are presented. For each type of laser or LED, the device physics is briefly discussed and suggestions are given on how to tune the input parameters in DESSIS to achieve various laser or LED output.

Users are referred to the examples in Section 25.2.1 on page 15.373 and Section 25.2.2 on page 15.378 for the generic format of the syntaxes in the command file, which is similar for simulating all laser diodes and LED types. Only the relevant portions of the syntax that are associated with differentiating features are described here.

The parameters for a laser or an LED simulation are contained in the `Physics-Laser` or `Physics-LED` sections of the command file. Table 15.158 lists the available options in these sections. Some options are only specific to a particular type of laser. For example, `LongitudinalModes`, `TransverseModes`, and `DFB` are *only* applicable to edge-emitting lasers; it does not make sense to use them in VCSEL or LED simulations.

Table 15.158 Feature keywords in the Physics-Laser or Physics-LED sections

Feature keyword	Parameter/Description	Reference
`Optics(<parameters>)`	`OptConfin=<float>`	Section 26.4 on page 15.390
	`FEScalar, FEVectorial, EffectiveIndex, TMM1D, RayTrace` (optical mode solvers)	Chapter 27 on page 15.399
`LongitudinalModes`	Specifies that longitudinal optical modes must be calculated.	Section 30.2 on page 15.491
`TransverseModes`	Specifies that transverse optical modes must be calculated (default).	
`DFB(DFBperiod=<float>)`	Switches on DFB feature where `<float>` is the period of the DFB grating [µm].	
`GroupRefract=<float>`	Specifies fixed effective index, ignoring the effective index computed by the optical solvers.	
`CavityLength=<float>`	Length of laser cavity [µm].	Section 26.3 on page 15.388
`lFacetReflectivity=<float>`	Left-facet power reflectivity.	
`rFacetReflectivity=<float>`	Right-facet power reflectivity.	
`OpticalLoss=<float>`	Background optical loss [1/cm].	
`SponEmiss=<float>`	Spontaneous emission factor.	

Table 15.158 Feature keywords in the Physics-Laser or Physics-LED sections

Feature keyword	Parameter/Description	Reference
`RefractiveIndex(<parameter>)`	`TemperatureDep`	Section 29.5 on page 15.477
	`CarrierDep`	
`VCSEL(<parameters>)`	Presence of this keyword signifies VCSEL simulation.	Section 30.3 on page 15.494
	`NearField(<float>,<float>,<int>)`	Section 27.10 on page 15.433
`QWTransport`	Activates 'three-point' QW model.	Section 28.2 on page 15.440
`QWExtension=AutoDetect`	Reads QW widths automatically.	
`QWScatModel`	Activates QW scattering model.	
`eScatTime=<float>`	Electron capture time in the QW [s].	
`hScatTime=<float>`	Hole capture time in the QW [s].	
`eQWMobility=<float>`	Mobility for bound state electrons [cm^2/Vs].	
`hQWMobility=<float>`	Mobility for bound state holes [cm^2/Vs].	
`QWShallow`	Activates shallow QW scattering model.	
`Strain`	Activates QW strain model.	Section 28.7 on page 15.451
`SplitOff=<float>`	Spin-orbit split-off energy [eV].	
`FreeCarr`	Switches on free carrier absorption.	Section 29.1 on page 15.471
`Broadening(<parameter>)`	Activates gain-broadening models or nonlinear gain saturation model.	Section 28.4 on page 15.446, Section 28.5 on page 15.447.
`SponScaling=<float>`	Scaling factor for matrix element of spontaneous emission.	Section 28.3 on page 15.443
`StimScaling=<float>`	Scaling factor for matrix element of stimulated emission.	
`BandStructure(<parameters>)`	`CrystalType=ZincBlende` (Activates k.p method for zinc-blend crystal lattice)	Section 28.9 on page 15.455
	`Order=<parameter>` (Choice of `Nokp`, `4x4kp`, `6x6kp`, and `8x8kp`. Default is `Nokp`.)	
	`NumkValues=<int>` (Number of discretization points for the k-space.)	
`Strain(RefLatticeConst=<float>)`	Input strain reference lattice constant for the k.p method.	
`ManyBodyEffects(Type=<parameter>)`	`FCT` (free carrier theory, default)	
	`SHF` (screen Hartree–Fock potential)	

30.2 Edge-emitting lasers

The 2D edge-emitting laser structure is essentially a waveguide problem (see Section 26.4 on page 15.390) in the transverse plane, with the cavity resonance occurring in the longitudinal direction. The main type of cavity resonance in the longitudinal direction used is the Fabry–Perot type where the resonant wavelength is chosen as the location of the gain peak. There is also an option to activate a simple distributed feedback (DFB) laser simulation. In addition, multiple transverse modes or multiple longitudinal modes can be simulated.

Two choices of optical mode solver are available for edge-emitting lasers: `FEScalar` and `FEVectorial`. These solvers have been described in Section 27.3 on page 15.401.

DESSIS–Laser offers two choices of multimode simulation: transverse and longitudinal modes. Such a simulation can be performed by setting the keyword `Modenumber=<int>` to greater than one, and specifying one of the keywords `TransverseModes` (which is the default and can be omitted) or `LongitudinalModes` in the `Physics-Laser` section of the command file:

```
Physics {...
   Laser (...
      Optics (
         FEVectorial (...)
      )
      # --- Choose transverse or longitudinal modes, but NOT both ---
      TransverseModes
#     LongitudinalModes
      ModeNumber = 5            # can choose up to 10 modes
      ...
   )
}
```

For each longitudinal or transverse mode, one photon rate equation is solved.

NOTE Users can select either multiple transverse modes or multiple longitudinal modes simulation, but not both simultaneously.

30.2.1 Multiple transverse modes

A maximum of ten transverse modes are allowed. Different transverse modes have different spatial distribution of the optical density. Therefore, each mode experiences different modal gains, giving rise to different stimulated gain spectra. Assuming a Fabry–Perot cavity, the wavelength of each mode is given by the location of the peaks of the corresponding gain spectrum. In a DFB simulation, the wavelength is set and the modal gain of each mode is taken as the value of its respective gain spectrum at this wavelength.

Depending on the current flow distribution, the carriers available to contribute to the material gain of each mode are determined by carrier transport. The different distributions of spatial optical intensity of different modes mean that the carriers are depleted (by stimulation recombination) at different locations, and this is commonly called the spatial hole-burning effect. The user can view this effect by plotting the carrier density distribution in the active region of the device.

30.2.2 Multiple longitudinal modes

The longitudinal mode simulation is only possible with Fabry–Perot-type cavities. If the keyword `LongitudinalModes` is set in the `Physics-Laser` section of the command file, a multilongitudinal mode calculation is performed. DESSIS automatically selects an odd number of modes, which ensures one central (or main) mode and an equal number of modes, smaller and greater than the main mode frequency. The frequency of the main mode is determined as in the single-mode case by calculating the Fabry–Perot mode with maximum mode gain.

The spacing of the frequencies of the side modes is calculated according to the resonant frequencies in the Fabry–Perot cavity, which is in turn defined by the cavity length (keyword `CavityLength` in the `Physics-Laser` section). There is only one transverse mode associated with the multiple longitudinal modes.

For 2D simulations, this means that all the longitudinal modes share the same modal gain spectrum and, hence, the same carrier population for stimulated recombination. The multiple longitudinal-mode simulation is important for analyzing the suppression ratio of the side mode of the laser.

NOTE The longitudinal mode option cannot be used with the DFB option.

30.2.3 Simple distributed feedback model

A simple distributed feedback (DFB) laser simulation can be performed by setting the keyword `DFB` in the `Physics-Laser` section of the command file:

```
Physics {...
   Laser (...
      Optics (
         FEVectorial (...)
      )
      # --- Specify DFB simulation
      TransverseModes
      DFB (Period=0.11)     # [microns]
      CavityLength = 200    # [microns]
      lFacetReflectivity = 0.3
      rFacetReflectivity = 0.3
      GroupRefract = 3.4        # fix effective index
      ...
   )
}
```

The argument `DFBPeriod=<float>` specifies the period of the DFB grating, Δ, in units of μm. As a result, the lasing wavelength is assumed to occur at the Bragg wavelength, $\lambda_B = 2n_{eff}\Delta$, where n_{eff} is the effective refractive index. The effective index is solved from the Helmholtz equation using either the scalar (`FEScalar`) or vectorial (`FEVectorial`) optical solvers and can change if the refractive index profile changes in the simulation. The wave propagation in the longitudinal DFB grating, however, is not simulated. The user can also set a fixed constant effective index with the keyword `GroupRefract=<float>`.

30.2.4 Bulk active-region edge-emitting lasers

Bulk active-region lasers can also be simulated. Similar to the case of the quantum well laser, users must specify the active material region by the keyword `Active`. In the `Physics-Laser` section of the command file, all the quantum well–related keywords should be removed: `QWTransport`, `QWExtension`, `QWScatModel`, `QWeScatTime`, `QWhScatTime`, `Strain`, and `SplitOff`. In this case, DESSIS–Laser treats the active region as a bulk region, and the stimulated gain is computed as a bulk material gain. The carriers are assumed to scatter into the bulk active region by thermionic emission.

30.2.5 Device physics and parameter tuning

There are many quantities that users may want to optimize for the best laser diode performance or may want to tune in order to match experimental results. A selected list of these quantities and ways to tune them are:

Optical mode shape	The shape is controlled by the geometry and refractive index of the device. GENESISe provides an automatic parameterization option for the user to optimize geometric feature sizes, layer thickness, refractive index profile, and so on for the required mode shape.
Discretization of optical grid	The optical solvers use the finite element method and a general rule is to use 20 points per wavelength to generate the mesh. This should provide an accurate solution for the optical mode.
Lasing threshold current	The threshold current is a function of the dark recombinations (Auger and SRH), optical losses, gain spectrum, and radiative recombination. The SRH lifetimes and Auger coefficients can be changed in the parameter file. Additional background optical loss can be input by `OpticalLoss=<float>` in the `Physics-Laser` section of the command file. The stimulated and spontaneous gain spectrum can be scaled by the keywords `StimScaling=<float>` and `SponScaling=<float>` in the `Physics-Laser` section of the command file.
Slope efficiency of L–I curve	This is primarily determined by the injection efficiency and the photon lifetime, which can be changed by the optical losses. However, changing the optical losses also affects the threshold current.
Temperature profile	The temperature distribution is sensitive to the boundary conditions specified at the thermodes. Users can change the `SurfaceResistance` in the `Thermode` section of the command file to tune the temperature profile (see Section 29.7 on page 15.481).
Thermal rollover	The rollover is caused by self-heating effects of the laser diode. Possible major causes are Auger recombination and increased quantum well current leakage.

30.2.6 Leaky waveguide lasers

In many laser designs, the refractive index of the substrate or top layers is near the value of the guiding layer and is higher than that of the cladding layers. In such a case, any waves penetrating into the substrate or top layer will radiate outwards contributing to additional losses of the mode. Such leakage can also be used to discriminate higher order modes (which have higher losses) from the fundamental mode so that single-mode high-power laser diodes can be designed.

The `FEVectorial` optical solver coupled with PML (see Section 27.5 on page 15.410) makes DESSIS an ideal tool to simulate such leaky waveguide lasers. In fact, DESSIS has been successfully used to optimize leaky waveguide lasers for single-mode high-power operations [195].

30.3 Vertical-cavity surface-emitting lasers

Vertical-cavity surface-emitting lasers (VCSELs) are difficult devices to simulate due to their many layers (usually more than 100 layers). A full 3D simulation of a VCSEL is not feasible because it requires an excessive amount of computing resources.

To reduce the computational load, cylindrical symmetry is assumed so that the VCSEL problem is reduced to a quasi-2.5-dimensional problem. Each mesh on the (ρ, z) plane represents a solid ring rotated around the z-axis. As a result of such symmetry, the optical field can be expanded in cylindrical harmonics, and this further helps to reduce the size of the problem (see Section 26.5 on page 15.392).

Three different types of optical solver are available to solve for the cavity modes of a VCSEL. There is one vectorial solver (`FEVectorial`) and two scalar solvers (`EffectiveIndex` and `TMM1D`). Only the vectorial solver provides accurate computation of the scattering and diffraction losses in a VCSEL of any type of geometry. This is important in computing the photon lifetime, which is a critical parameter in determining the slope efficiency of the light power output and the threshold current.

Nevertheless, the scalar solvers are extremely fast and most useful in the initial design phase of the VCSEL structure. In particular, the effective index method (keyword `EffectiveIndex`) can compute fairly accurate resonant wavelengths for strongly index-guided VCSELs, including oxide-confined VCSELs.

All VCSEL simulations in DESSIS must be performed on two different grids: one for the electrical problem and one for the optical problem.

30.3.1 Different grid and structure for electrical and optical problems

A typical VCSEL structure contains over 100 layers. DESSIS can manage such a large electrical problem, but the resultant Jacobian matrix may be too large for standard computers to handle. The carrier transport behavior in the distributed Bragg reflectors (DBRs) is not interesting with regard to the laser physics, and the DBRs will probably contribute only to an additional series resistance. Therefore, the size of the electrical problem can be simplified and reduced by replacing the DBR layers with an equivalent homogeneous bulk material with similar total resistance and thermal conduction properties. However, the actual layered structure must be used for the optical simulation because the optical resonance depends on the exact geometry of the VCSEL cavity.

Therefore, there will be two different structures and grids for the electrical and optical problems. To handle this, the dual-grid mixed-mode capability of DESSIS is used. The syntax for such a dual-grid simulation is described in Section 30.3.4 on page 15.496.

For the vectorial optical (`FEVectorial`) solver, the optical mesh is discretized according to the requirements of the finite element method. Adaptive meshing can be used to refine the mesh at sharp corners where scattering and diffraction effects are strong. For the scalar optical solvers (`EffectiveIndex` and `TMM1D`), the meshing of the optical grid can be relaxed: users will draw the VCSEL structure and apply coarse meshing to generate the optical grid.

30.3.2 Aligning resonant wavelength within the gain spectrum

Besides being a cavity problem, a VCSEL is different from an edge-emitting laser mainly in the size of the gain volume. The gain volume in a VCSEL is many times smaller than that of an edge-emitting laser. Therefore, ensuring a VCSEL lases is an intricate design task of minimizing the source of optical and electrical losses and maximizing the gain. One critical issue in VCSEL design is to align the resonant wavelength such that it is within the gain spectrum during lasing.

Self-heating of the VCSEL cavity causes red-shifts in both the gain spectrum (band-gap reduction as shown in Figure 15.113) and the resonant wavelength (thermal lensing effect). However, the shift in the gain spectrum is many times that of the resonant wavelength. Therefore, it is important to adjust the resonant wavelength such that it is near the peak of the gain spectrum at the required operating bias.

Figure 15.114 shows two starting positions of the resonant mode. As the bias increases, the gain spectrum shifts left, and the resonant mode of the model in Figure 15.114 (*right*) runs the risk of not obtaining enough gain required for lasing. Therefore, it becomes a design issue of geometry and quantum well material gain to ensure that the resonant wavelength and gain spectrum (before lasing) resembles the model in Figure 15.114 (*left*).

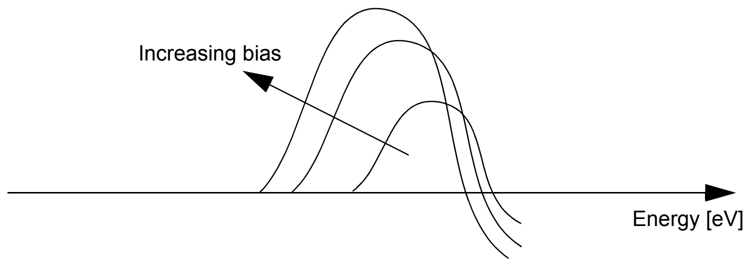


Figure 15.113 As bias increases, the gain spectrum red-shifts (band-gap reduction) due to self-heating effects

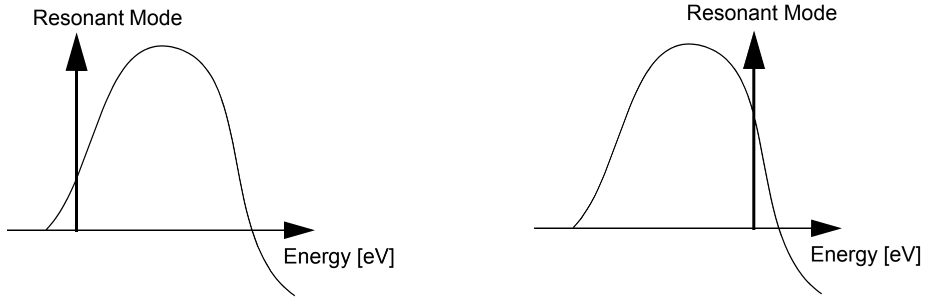


Figure 15.114 Resonant wavelength chosen to with different starting positions from the gain spectrum at very low bias (not lasing yet)

30.3.3 Device physics and parameter tuning

For a VCSEL, many quantities such as threshold current, slope efficiency, and stimulated and spontaneous gain can be similarly tuned as in the case of an edge-emitting laser. Other quantities that are of interest in a VCSEL simulation are:

Resonant wavelength selection
: The resonant wavelength can be changed by changing the thickness or refractive indices of the DBR layers or the lambda cavity of the VCSEL. Users can use the optics stand-alone option and the `EffectiveIndex` scalar solver to accomplish this optical design problem efficiently.

Aligning wavelength with gain spectrum
: It is easier to tune the resonant wavelength rather than the gain spectrum. Users are advised to run the simulation once to look at the gain spectrum shifts, then change the resonant wavelength as described in the previous paragraph.

Gain spectrum shifts
: The band gap is reduced as temperature increases (due to self-heating). Higher carrier densities also result in band-gap renormalization caused by many-body effects. However, the band-gap renormalization shift is very small compared to the temperature band-gap shift.

Scattering and diffraction losses
: The optical losses give the photon lifetime which is a critical parameter in the threshold current and slope efficiency. Only the vectorial optical (`FEVectorial`) solver can compute the optical losses accurately. If users plan to use the scalar optical solvers (`EffectiveIndex` or `TMM1D`), it is advisable to run the vectorial optical solver once to obtain the accurate optical losses, then append an appropriate background loss (using keyword `OpticalLoss` in `Physics-Laser` section) or diffraction loss (using keyword `DiffractionLoss` in `EffectiveIndex` section) to the scalar optical solvers to enhance the accuracy of the scalar simulation.

30.3.4 Example syntax for VCSEL simulation

A VCSEL simulation is only performed if the keyword `VCSEL` is included in the `Physics-Laser` section of the command file:

```
# ----- Dessis command file for dual grid VCSEL simulation -----
File {
    Output = "dual_log"
}

# ----- Control of the numerical method in the mixed-mode circuit -----
Math {
   NoAutomaticCircuitContact
   DirectCurrentComputation
   Method = blocked
   Submethod = pardiso
   Digits = 5
   Extrapolate
   ErReff(electron) = 1.e3
   ErReff(hole) = 1.e3
   Iterations = 30
   Notdamped = 50
   RelErrControl
   ElementEdgeCurrent
```

```
}

# ===== Define the Optical grid =====
# ----- Use keyword OpticalDevice -----
OpticalDevice optgrid {

   File {
      # ----- Read in the optical mesh -----
      Grid = "optmesh_mdr.grd"
      Doping = "optmesh_mdr.dat"
      Parameters = "des_las.par"
   }
   Plot {
      LaserIntensity
      OpticalIntensityMode0
   }
   # ----- Material region physics for the optical problem -----
   ... # all the layers including the DBR layers

   # ----- Quantum wells -----
   Physics (region="qw1") {
      MoleFraction(xfraction = 0.8 Grading=0.00)
      Active
   }
   Physics (region="barr") { MoleFraction(xfraction=0.09) }
   Physics (region="qw2") {
      MoleFraction(xfraction = 0.8 Grading=0.00)
      Active
   }

}
# ===== End of Optical grid definition =====

# ===== Define the electrical grid and solver info =====
# ----- Use keyword Dessis -----
Dessis electricaldev {

   Electrode {
      { Name="p_Contact" voltage=0.8 AreaFactor=1 }
      { Name="n_Contact" voltage=0.0 AreaFactor=1 }
   }
   File {
      # ----- Read in electrical grid mesh -----
      Grid = "elecmesh_mdr.grd"
      Doping = "elecmesh_mdr.dat"
      Parameters = "des_las.par"

      Current = "elec_current"
      Plot =    "elec_plot"
      SaveOptField = "laserfield"
      ModeGain = "gain"
      VCSELNearField = "nf"
      OptFarField = "far"
   }
   Plot {
      # ----- Can include a long list -----
      LaserIntensity
      OpticalIntensityMode0
   }
   Physics {
      AreaFactor = 1
      # ------ Laser definition ------
      Laser(
         Optics(
                FEVectorial(EquationType = Cavity
                            Coordinates = Cylindrical
                            TargetWavelength = (781.2 776.5)    # initial guesses [nm]
                            TargetLifetime = (2.4 1.2)          # initial guesses [ps]
                            AzimuthalExpansion = (1 0)          # cylindrical harmonic order
                            ModeNumber = 2
                )
         )
         # ----- Signify this is a VCSEL simulation -----
         VCSEL( NearField(10.0,0,100) ) # the argument gives the near field parameters
         OpticalLoss = 10.0             # [1/cm]
         # ----- Choose Gain broadening -----
         Broadening (Type=Lorentzian Gamma=0.10)        # [eV]
         # ----- Specify QW parameters -----
         qwTransport
         qwExtension = AutoDetect       # auto read QW widths
         qwScatmodel
         QWeScatTime = 8e-13            # [s]
         QWhScatTime = 4e-13            # [s]
         eQWMobility = 9200             # [cm^2/Vs]
         hQWMobility = 400              # [cm^2/Vs]
         # ----- QW Strain effects -----
         Strain
         SplitOff = 0.34                # [eV]
         # ----- can scale stim and spon gain independently -----
         StimScaling = 1.0
         SponScaling = 1.0
         # ----- Specify dependency of refractive index ----
         RefractiveIndex(TemperatureDep CarrierDep)
      )
      # ----- Specify transport physics -----
      Thermionic
      HeteroInterfaces
      Mobility ( DopingDep )
      Recombination ( SRH Auger )
      EffectiveIntrinsicDensity ( NoBandGapNarrowing )
      Fermi
   }
   # ----- Material region physics for electrical problem -----
   # --- Simplify the DBR layers by an equivalent bulk region ---
   Physics (region="pDBR_equivalent") { MoleFraction(xfraction=0.35) }
   Physics (region="nDBR_equivalent") { MoleFraction(xfraction=0.35) }
   Physics (region="psch") { MoleFraction(xfraction=0.09) }
   Physics (region="nsch") { MoleFraction(xfraction=0.09) }
   # ----- Quantum wells -----
   Physics (region="qw1") {
      MoleFraction(xfraction = 0.8 Grading=0.00)
      Active
   }
   Physics (region="barr") { MoleFraction(xfraction=0.09) }
   Physics (region="qw2") {
      MoleFraction(xfraction = 0.8 Grading=0.00)
      Active
   }

   # ----- Control the numerical method in the electrical problem -----
   Math {
      Digits = 7
      ElementEdgeCurrent
      # ----- Need to include this to specify cylindrical symmetry -----
      Cylindrical
   }
```

```
}
# ===== End of electrical grid definition =====

# ===== Define the circuit mixed-mode system =====
System {
   # ----- Define opt1 of type optgrid -----
   optgrid opt1 ()
   # ----- Define d1 of type electricaldev, and coupled to opt1 -----
   electricaldev d1 (p_Contact=vdd n_Contact=gnd) {Physics{OptSolver="opt1"}}
   # ----- Set the initial bias voltage to circuit contacts -----
   Vsource_pset drive(vdd gnd){ dc=0.8 }
   Set ( gnd=0.0 )
}

# ===== Solver part =====
Solve {
   Poisson
   coupled { Hole Electron Poisson Contact Circuit }
   coupled { Hole Electron QWhScatter QWeScatter Poisson Contact Circuit }
   coupled { Hole Electron QWhScatter QWeScatter Poisson Contact Circuit PhotonRate}

   quasistationary (
        InitialStep = 0.001
        MaxStep = 0.01
        Minstep = 1e-7
        # ----- Plot and save various quantities at specified intervals of bias -----
        Plot { range=(0,1) intervals=5 }
        PlotGain { range=(0,1) intervals=5 }
        PlotGainPara { range=(1.22,1.32) intervals=150 }
        SaveOptField { range=(0,1) intervals=3 }
        VCSELNearField { range=(0,1) intervals=3 }
        PlotFarField { range=(0,1) intervals=2 }
        PlotFarFieldPara { range=(80,80) intervals=50 Scalar1D Vector2D}
        # ----- Specify final voltage -----
        Goal { Parameter=drive.dc Value=1.6 })
        {
           Plugin (BreakOnFailure) {
                Coupled { Electron Hole Poisson Contact Circuit
                           QWeScatter QWhScatter PhotonRate }
                Optics
                Wavelength
           }
        }
}
```

Compare this VCSEL example with the edge-emitting laser example in Section 25.2.2 on page 15.378, to highlight the syntaxes required for a VCSEL simulation. Some comments about the above example are:

- The vectorial optical solver (`FEVectorial`) has been used. To use the scalar solvers, users need only replace the `FEVectorial` section with the `EffectiveIndex` or `TMM1D` section.
- Dual-grid mixed-mode simulation is required for VCSELs regardless of whether the vectorial or scalar optical solvers are chosen.
- In the `Math` section for the electrical device, the keyword `Cylindrical` must be included so that the mesh volumes and areas in the finite box integration method are computed with cylindrical symmetry in mind.

30.4 Light-emitting diodes

From an electronic perspective, light-emitting diodes (LEDs) are similar to lasers operating below the lasing threshold. Consequently, the electronic model contains similar electrothermal parts and quantum-well physics as in the case of a laser simulation.

The optics are different because the field is not confined to any waveguide or cavity modes, but radiates incoherently from the device. The only optical solver available for LED simulation is `RayTrace` (see Section 27.8 on page 15.419). In DESSIS, raytracing calculates the optical intensity in the device and the extraction efficiency (fraction of rays that radiate outwards from the device). The other results of an LED simulation are the total spontaneous power output, voltage, current, and the mean wavelength. These values are plotted in the current file (keyword `Current="String"` in the `File` section).

30.4.1 Single-grid versus dual-grid LED simulation

Both single-grid and dual-grid LED simulations are possible. However, in the case of a single-grid simulation, raytracing takes a longer time for the following reason: raytracing builds a binary tree for each starting ray. Each branch of the tree corresponds to a ray at a mesh cell boundary. If the materials in two adjoining cells are different, the ray splits into refracted and reflected rays, creating two new branches. If the materials are the same in adjoining cells, the propagated ray creates a new branch. A fine mesh increases the depth of the branching significantly. Each new branch of the binary tree is created dynamically and if dynamic memory allocation of the machine is not fast enough, the tree creation of the raytracing becomes a bottleneck in the simulation.

To overcome this problem, the grids for the electrical problem and raytracing problem are separated. The optical grid for raytracing is meshed coarsely. The ideal and coarsest possible optical mesh would have each material region as a single mesh cell. The binary tree created will then be small and raytracing is more efficient. Such a coarse mesh enables users to compute the extraction efficiency and output radiation pattern. However, the optical intensity within the device cannot be resolved with such a coarse mesh.

NOTE Due to the random nature of spontaneous emission, raytracing and the electronic solver cannot be coupled self-consistently.

30.4.2 LED output power

The LED output power computed in DESSIS is the integral sum of the spontaneous emission power at every energy interval in the spontaneous gain spectrum (see (Eq. 15.555)). This output power, together with the extraction efficiency, is saved as a function of bias in the current file (specified by `Current="string"` in the `File` section) at the end of the simulation. The total measured power of the LED should be the product of these two quantities, that is:

$$P_{measured} = P_{total}^{sp} \times \text{Extraction Efficiency} \qquad (15.616)$$

In addition, the bias currents and voltages are available in the current file. Therefore, the user can easily compute the internal quantum efficiency and the wallplug efficiency.

An average wavelength for the spontaneous emission output is computed by averaging the wavelengths of the peak spontaneous gain at every active vertex.

30.4.3 Device physics and tuning parameters

Unlike a laser diode, an LED does not have a threshold current. Therefore, the carriers in the active region are not limited to any threshold value. This means that the spontaneous gain spectrum continues to grow as the bias current increases. The limiting factor for the growth is when the quantum well active region is completely filled and leakage current increases significantly.

There are two main design concerns for an LED:

- Extraction efficiency. This is mainly a problem of the geometric shape of the LED structure. Many LED structures have tapered sidewalls to help couple more light out of the device. The slope of the taper can be set as a parameter using GENESISe, and the automatic parameter variation feature can be used to optimize the extraction efficiency of the LED geometry.
- Uniform current spreading. It is desirable to spread the current uniformly across the entire active region so that total spontaneous emissions can be increased. In DESSIS, there is an option to switch off raytracing in an LED simulation. Switching off raytracing only forgoes the extraction efficiency and radiation pattern computation; the total spontaneous emission power is still calculated. This can assist users in the faster optimization of the LED device for uniform current spreading.

30.4.4 Example syntax for LED simulation

An LED simulation is activated by the keyword `LED` in the `Physics` section of the command file:

```
# ----- Dessis command file for dual grid LED simulation -----
File {
    Output = "dual_log"
}

# ----- Control of the numerical method in the mixed-mode circuit -----
Math {
   NoAutomaticCircuitContact
   DirectCurrentComputation
   Method = blocked
   Submethod = pardiso
   Digits = 7
   Extrapolate
   ErReff(electron) = 1.e3
   ErReff(hole) = 1.e3
   Iterations = 30
   Notdamped = 50
   RelErrControl
   ElementEdgeCurrent
}

# ===== Define the Optical grid =====
# ----- Use keyword OpticalDevice -----
OpticalDevice optgrid {

   File {
      # ----- Read in the optical mesh -----
      Grid = "optmesh_mdr.grd"
      Doping = "optmesh_mdr.dat"
      Parameters = "des_las.par"
   }
   Plot {
      OpticalIntensityMode0
   }
   # ----- Material region physics for the optical problem -----
   Physics (region="pbulk") { MoleFraction(xfraction=0.35) }
   Physics (region="nbulk") { MoleFraction(xfraction=0.35) }
   Physics (region="psch") { MoleFraction(xfraction=0.09) }
   Physics (region="nsch") { MoleFraction(xfraction=0.09) }
   # ----- Quantum wells -----
   Physics (region="qw1") {
      MoleFraction(xfraction = 0.8 Grading=0.00)
      Active
   }
   Physics (region="barr") { MoleFraction(xfraction=0.09) }
   Physics (region="qw2") {
      MoleFraction(xfraction = 0.8 Grading=0.00)
      Active
   }

}
# ===== End of Optical grid definition =====

# ===== Define the electrical grid and solver info =====
# ----- Use keyword Dessis -----
Dessis electricaldev {

   Electrode {
      { Name="p_Contact" voltage=0.8 AreaFactor=1 }
      { Name="n_Contact" voltage=0.0 AreaFactor=1 }
   }
   File {
      # ----- Read in electrical grid mesh -----
      Grid = "elecmesh_mdr.grd"
      Doping = "elecmesh_mdr.dat"
      Parameters = "des_las.par"

      Current = "elec_current"
      Plot =    "elec_plot"
      SaveOptField = "ledfield"
      ModeGain = "gain"
      LEDRadiation = "rad"
   }
   Plot {
      # ----- Can include a long list -----
      OpticalIntensityMode0
   }

   Physics {
      AreaFactor = 2    # for symmetric devices
      # ----- Activate LED simulation -----
      LED (
         Optics (
            # ----- Choose ray tracing to compute extraction efficiency -----
            RayTrace(
                # ----- Info about LED structure -----
                Symmetry = Symmetric           # or NonSymmetric
                Coordinates = Cartesian        # or Cylindrical
                # ----- Specify absorption and refractive index models -----
                SemAbsorption ( model = ODB )
                RefractiveIndex( model = ODB )
                # ----- Specify Starting rays parameters -----
                RaysPerVertex = 40              # Number of starting rays per active vertex source
                RaysRandomOffset                # Randomize starting ray angle
                # ----- Specify ray trace terminating conditions -----
                DepthLimit = 10                 # finish after ray crosses 10 material boundaries
                MinIntensity = 1e-7             # finish if ray intensity is less than 1e-7

                LEDRadiationPara(1000.0,180)    # (<radius-microns>, Npoints)
                # ----- Auxiliary features of LED ray tracing -----
#               Disable                         # disable ray tracing but still run the LED simulation
#               Print                           # print out all the rays in a grid file
            )
        )

        # ----- Other parameters of the LED structure -----
        Cavitylength = 200                      # for 2D simulation [microns]

        # ----- Choose spontaneous gain broadening -----
        Broadening (Type=Lorentzian Gamma=0.01)

        # ----- Specify QW Physics -----
        QWTransport
        QWExtension = AutoDetect                # auto read QW widths
        QWScatModel
        QWeScatTime = 1e-13                     # [s]
        QWhScatTime = 2e-14                     # [s]
        eQWMobility = 9200                      # [cm^2/Vs]
        hQWMobility = 400                       # [cm^2/Vs]
        # ----- QW strain effects -----
        Strain
        SplitOff = 0.34                         # [eV]
        # ----- Can scale spon gain independently -----
        SponScaling = 1.0
     )

     # ----- User specified physics of transport -----
     Thermionic
     HeteroInterfaces
     Mobility ( DopingDep )
     Recombination ( SRH Auger )
     EffectiveIntrinsicDensity ( NoBandGapNarrowing )
     Fermi
  }

  # ----- Material region physics for electrical problem -----
  Physics (region="pbulk") { MoleFraction(xfraction=0.35) }
  Physics (region="nbulk") { MoleFraction(xfraction=0.35) }
  Physics (region="psch") { MoleFraction(xfraction=0.09) }
  Physics (region="nsch") { MoleFraction(xfraction=0.09) }
  # ----- Quantum wells -----
  Physics (region="qw1") {
     MoleFraction(xfraction = 0.8 Grading=0.00)
     Active
  }
  Physics (region="barr") { MoleFraction(xfraction=0.09) }
  Physics (region="qw2") {
     MoleFraction(xfraction = 0.8 Grading=0.00)
     Active
  }

  # ----- Control the numerical method in the electrical problem -----
  Math {
     Digits = 5
     ElementEdgeCurrent
```

```
      # ----- Need to include this if cylindrical symmetry is chosen -----
#     Cylindrical
   }
}
# ===== End of electrical grid definition =====

# ===== Define the circuit mixed-mode system =====
System {
   # ----- Define opt1 of type optgrid -----
   optgrid opt1 ()
   # ----- Define d1 of type electricaldev, and coupled to opt1 -----
   electricaldev d1 (p_Contact=vdd n_Contact=gnd) {Physics{OptSolver="opt1"}}
   # ----- Set the initial bias voltage to circuit contacts -----
   Vsource_pset drive(vdd gnd){ dc=0.8 }
   Set ( gnd=0.0 )
}

# ===== Solver part =====
Solve {
   Poisson
   coupled { Hole Electron Poisson Contact Circuit }
   coupled { Hole Electron QWhScatter QWeScatter Poisson Contact Circuit }
   coupled { Hole Electron QWhScatter QWeScatter Poisson Contact Circuit PhotonRate}

   quasistationary (
        InitialStep = 0.001
        MaxStep = 0.01
        Minstep = 1e-7
        # ----- Plot and save various quantities at specified intervals of bias -----
        Plot { range=(0,1) intervals=5 }
        PlotGain { range=(0,1) intervals=5 }
        PlotGainPara { range=(1.22,1.32) intervals=150 }
        SaveOptField { range=(0,1) intervals=3 }
        PlotLEDRadiation { range=(0,1) intervals=2 }
        # ----- Specify final voltage -----
        Goal { Parameter=drive.dc Value=1.6 })
        {
           Plugin (BreakOnFailure) {
                Coupled { Electron Hole Poisson Contact Circuit
                           QWeScatter QWhScatter PhotonRate }
                Wavelength
           }
        }
}
```

Some comments about the above syntax are:

- The dual-grid electrical and optical feature has been used. If users select a single-grid simulation, they only need to copy the `Physics-LED` section of this example and insert it into the `Physics` section of a typical single-grid command file such as in Section 25.2.1 on page 15.373.
- In the `Solve` section, the keyword `PhotonRate` must be added to the `Coupled` statement. The LED simulation requires the interface of the photon rate equation to access the spontaneous emission gain calculations. However, the photon rate equation is not actually solved.

Part IV Mesh and Numeric Methods

This part of the DESSIS manual contains the following chapters:

CHAPTER 31 Automatic grid generation and adaptation module AGM

31.1 Overview

DESSIS supports the automatic generation and adaptation of 2D quadtree-based simulation grids for physical devices. The approach is based on a local anisotropic grid adaptation technique for the drift-diffusion model [163][164] and has been formally extended to thermodynamic and hydrodynamic simulations.

NOTE In its current status, AGM is not designed to improve the speed of simulations but rather to control the accuracy of the solution. In fact, using AGM slows down the simulation time considerably as the control of the grid sizes is difficult and the recomputation of solutions on adaptively generated grids is, in the presence of strong nonlinearities, a time-consuming task. Therefore, it is not recommended to use AGM throughout large simulation projects in a fully automatic adaptation modus. AGM in its current status may be used in a semi-automatic fashion to understand which parts of a device grid require more refinement to improve solution accuracies and to construct for a set of simulations appropriately fixed simulation grid(s). Its integration in DESSIS is incomplete as incompatibilities with certain DESSIS features occur.

The accuracy of approximate solutions computed by many simulation tools depends strongly on the simulation grid used in the discretization of the underlying problem. The major aim of grid adaptation is to obtain numeric solutions with a controlled accuracy tolerance using a minimal amount of computer resources using *a posteriori* error indicators to construct appropriate simulation grids. The main building blocks of a local grid adaptation module for stationary problems are the adaptation criteria (local error indicators that somehow determine the quality of grid elements), the adaptation scheme (determining if and how the grid will be modified on the basis of the adaptation criteria), and the recomputation procedure of the approximate solution on adaptively generated grids. In the framework of finite element methods for linear, scalar elliptic boundary value problems grid adaptation has reached a mature status [165].

The semiconductor device problem consists of a nonlinearly coupled system of partial differential equations and the true solution of the problem exhibits layer behavior and singularities, posing additional difficulties for grid adaptation modules. Several adaptation criteria have been proposed in the literature [163]. For the overall robustness of adaptive simulations, the recomputation of the solution is a very serious (and, sometimes, very time-consuming) problem.

The adaptation procedure used in DESSIS is based on the approach developed in [163] and [164]. It uses the idea of equidistributing local dissipation rate errors and aims at accurate computations of the terminal currents of the device. The quadtree mesh structure of MESH is used to enable anisotropic grid adaptation on boundary Delaunay meshes required by the discretization used in DESSIS. The recomputation procedure relies on local and global characterizations of dominating nonlinearities and includes relaxation techniques based on the solution of local boundary value problems and a global homotopy technique for large avalanche generation.

31.1.1 Adaptation procedure

The adaptation flow is sketched in the following pseudo-code:

```
compute solution on actual grid(s)

coupled adaptation loop{
       // adaptation decision
       for all adaptive devices {
          compute adaptation criteria (including h/2-grid computations)
          check if adaptation is required
       }
       check if coupled adaptation is required

       // adaptation strategy
       for all adaptive devices {
          // grid adaptation
          criteria adaptation (tree loop)
          (directed) neighbor size refinement
          new simulation grid (conforming, delaunization)

          // device level data smoothing
          data interpolation (initial guess)
          electrostatic potential correction (EPC)
          nonlinear node block Jacobi smoothing (NBJI)
          avalanche homotopy
       }

       // system level smoothing
       Newton iteration to achieve self consistent solution of the fully coupled system
}
```

The coupled adaptation loop is iterated until the system fulfills the adaptation criteria or the maximal number of iterations is reached (`MaxCLoops`).

31.1.2 Adaptation decision

The decision as to whether the grid of a device must be updated depends on the (global) relative error of the observables F specified for the adaptation criteria. Refinement and coarsening adaptation are distinguished depending on the size of the relative error, that is, the grid will be refined if for one global error estimate $\eta(F)$ of the (Dirichlet or residual) criteria:

$$\eta(F) > \varepsilon_R(\varepsilon_A + |F^h|) \tag{15.617}$$

holds, where ε_R and ε_A are the relative and absolute error (specified by `RelError` and `AbsError`), respectively, and F^h is the actual value of the integral quantity (h is the mesh size parameter).

A coarsening adaptation occurs if all criteria satisfy:

$$\eta(F) \le \frac{1}{20}\varepsilon_R(\varepsilon_A + |F^h|) \tag{15.618}$$

31.1.3 Adaptation strategy

If the grid must be adapted, first, the adaptation criteria are applied to the elements of the grid tree performing one or several levels of anisotropic refinement and/or coarsening (tree adaptation loop or RCLoop). To make the mesh regular, a directed (anisotropic) neighbor size refinement is performed and a final delaunization step generates the new simulation mesh. Even in refinement adaptation, coarsening is allowed (if not disabled by the user) for elements with small errors. In a coarsening adaptation, no refinement is performed.

31.1.4 Adaptation criteria

The adaptation criteria determine if and how the grid will be modified. In [163], adaptation criteria for the nonlinearly coupled system of equations for the drift-diffusion model have been proposed, aiming at accurate computations of the terminal currents of the device. They use the close relationship between the system dissipation rate and the terminal currents, and estimate the error of the dissipation rate. This is performed by either solving related local Dirichlet problems or using the residual error estimation technique. Both techniques are well known in the framework of finite element discretizations for scalar elliptic boundary value problems [165]. The following adaptation criteria are supported by DESSIS.

31.1.4.1 Adaptation criteria based on local Dirichlet problems

These criteria construct locally refined meshes and solve local Dirichlet problems to estimate the error of the interesting integral quantity (criterion type `Dirichlet`). Their computation is quite expensive as they require the solution of additional problems, that is, they are based on implicit error indicators.The Dirichlet adaptation criteria estimate the *error of the quantity* F per grid element T as:

$$\eta_T(F) = \int_T \left|(\omega_F^h - \omega_F^{h/2})\right| dx \tag{15.619}$$

where ω_F^h and $\omega_F^{h/2}$ are the quantity density on the actual simulation grid and the locally refined h/2-grid, respectively. A weaker form of the error is given by the expression:

$$\eta_T^D(F) = \int_T (\omega_F^h - \omega_F^{h/2}) dx \tag{15.620}$$

and is called *deviation of the quantity* F on element T.

Dirichlet-type adaptation criteria are implemented for the so-called AGM dissipation rate, which is a weighted form of the physical system dissipation rate and the domain integral current (of a given contact). The AGM dissipation rate (`AGMDissipationRate`) is given as:

$$D_{AGM} = \hat{w}_n \int_\Omega \mu_n n |j_n|^2 dx + \hat{w}_p \int_\Omega \mu_p p |j_p|^2 dx + \hat{w}_r k_B T_l \int_\Omega R_{abs} \left|\log\left(\frac{np}{n_{i,\mathit{eff}}\, p_{i,\mathit{eff}}}\right)\right| dx \tag{15.621}$$

where $R_{abs} = \sum \hat{w}_{R_i} |R_i|$ is the weighted absolute sum of individual generation–recombination processes with their individual weights $\hat{w}_{R_i}$, and T_l is the lattice temperature (all weights $\hat{w}_i$ can be modified by the user).

The (electron) domain integral current at contact C is given as:

$$I_n^C = -\int_\Omega R h_n^C + j_n \nabla h_n^C dx \tag{15.622}$$

where h_n^C is a suitable test function with $h_n^C(x) = 1$ for locations x on contact C, $h_n^C(x) = 0$ for all locations x on contacts $C' \neq C$. The sum of electron and corresponding hole domain integral currents give the (total) domain integral current (`DomainIntegralCurrent`) at the specified contact.

31.1.4.2 Residual adaptation criteria

The residual adaptation criteria (so far, only available for the `AGMDissipationRate`) are explicit error estimators as they refer only to the actual solution (criterion type `Residual`). In analogy to standard methods, they measure jumps of the density of interest across inter-element boundaries.The error for element T is estimated as:

$$\eta_T(F) = \frac{|T|}{|N_e(T)|} \sum_{T' \in N_e(T)} \left|\omega_F^h(T') - \omega_F^h(T)\right| \tag{15.623}$$

where $\omega_F^h(T)$ denotes the approximate element functional density (on element T and T', respectively), $N_e(T)$ is the set of (semiconductor) elements sharing an edge with T, $|N_e(T)|$ is the number of elements, and $|T|$ is the volume of T.

31.1.4.3 Adaptation criteria based on element variation

Extending the adaptation criteria of [163], an adaptation criterion based on the variation of arbitrary (on vertices defined) functionals has been implemented (criterion type `Element`). The criterion uses simply the differences of vertex-based data f as an error indicator, that is:

$$\eta_T(f) = \max\{|f(x_i) - f(x_j)| : \overline{x_i x_j} \text{ edge of } T\} \tag{15.624}$$

Elements are refined if the value exceeds a user-specified value (using `MaxTransDiff`).

31.1.5 Solution recomputation

For the recomputation of the solution, the data is interpolated onto the new simulation, and iterative smoothing techniques are applied to improve the robustness of the procedure.

31.1.5.1 Device level data smoothing

In the first step, the electrostatic potential is adjusted to interpolated data by applying the so-called electrostatic potential correction (EPC), that is, a mixed linear and nonlinear Poisson equation is solved using a Newton algorithm. To achieve almost self-consistent solutions for the coupled equations of the device, local Dirichlet problems are solved approximately, resulting in the so-called nonlinear NBJI (node block Jacobi iteration). The node block iterations are performed only on a subset of all grid vertices.

For remarkable avalanche generation, a homotopy technique (or continuation technique), here called avalanche homotopy, is applied to improve the robustness of the recomputation procedure, that is, the true avalanche generation is globally decoupled from the equations and is integrated stepwise into the solution process. As the avalanche generation F_{ava} is an additive term, the equations to be solved $F(x) = F_b(x) + F_{\text{ava}}(x) = 0$ can be split into a basic part F_b and the avalanche generation term F_{ava}, where x represents the unknown solution variables.

With the fix avalanche generation $\widehat{F}_{\text{ava}}$ interpolated from the old grid, the avalanche homotopy now reads:

$$H_{\text{ava}}(x, t) = F_b(x) + (1 - t)\widehat{F}_{\text{ava}} + tF_{\text{ava}}(x) \tag{15.625}$$

where $t \in [0;1]$ is the homotopy parameter ramped from 0 to 1. For $t = 0$, the homotopy is reduced to a simplified problem, while for $t = 1$ the fully coupled system is solved. Thus, the avalanche homotopy is similar to a quasistationary simulation where the avalanche generation is ramped (instead of specified parameters).

31.1.5.2 System level data smoothing

On the system level, a self-consistent solution for the original fully coupled system of equations is computed by the Newton algorithm.

31.2 Adaptive device instances

A device instance is adaptive if the keyword `GridAdaptation` is specified as a section of the instance description. Several parameters can be passed to the instance by specifying parameter or body entries for the keyword, that is:

```
GridAdaptation ( <agm-device-par-list> ) { <agm-device-body-list> }
```

For adaptive devices, a mesh description from the MESH command and boundary files is necessary.

31.2.1 AGM device parameters

The possible parameter entries modify device-specific quantities.

Parameters affecting grid generation

The specification `MaxNumberMacroElements=<int>` limits the number of leaf elements in the quadtree (before possible neighbor size refinement and delaunization), that is, no refinement adaptation is performed if the specified value is exceeded.

`MaxCLoops` specifies the maximal number of adaptations of the instance in coupled adaptation loops.

`Weights` modifies the AGM dissipation rate (see (Eq. 15.621)) of the instance used in the adaptation criteria. An example is:

```
Weights ( eCurrent = 1. hCurrent = 1.e2 Recombination = 1.e3 Avalanche = 1.e-3 )
```

where `eCurrent`, `hCurrent`, and `Recombination` refer to the weights $\hat{w}_n$, $\hat{w}_p$, and $\hat{w}_r$, respectively; while `Avalanche`, for example, refers to the corresponding weight of the avalanche generation in R_{abs}. By default, all weights are 1.

The neighbor size refinement can be switched on and off by `[-] NeighborSizeRefinement`, and the material group–dependent ratios can be modified explicitly by using `NeighborSizeRatio`. The material group is `Semiconductor`, or `Insulator`, or `Conductor`.

For the Dirichlet adaptation criteria requiring solutions on locally refined meshes, two modes are available, namely, the element and the global patch mode. The element patch mode computes for each element a solution of local h/2-grid Dirichlet problems; the global patch mode computes the global solution on the h/2-grid and uses this in the adaptation criteria as a reference solution.

In the `RCLoop`, the tree adaptation loop is specified. The number of iterations applied to the element tree can be specified by using `Iterations = <int>` (default is 2), and by using `[-]Coarsening` if coarsening is allowed in refinement adaptations (default is enabled).

Table 15.159 AGM device parameter entries

Keyword syntax	Description	Default
`MaxNumberMacroElements = <int>`	Maximal number of leaf elements in macro element tree.	100 000
`MaxCLoops = <int>`	Maximal number of iterations per adaptive coupled system.	100 000
`Weights (<weight-list>)`	Weights in AGM dissipation rate.	1.
`[-] NeighborSizeRefinement`	Enables/disables neighbor size refinement.	enabled
`NeighborSizeRatio(<matgroup>) = <real>`	Allowed anisotropic edge length ratios for neighboring elements.	3. for semiconductor, 10^6 else
`PatchMode = Element | Global`	Affects only Dirichlet adaptation criteria: solve mode on h/2-grid for reference solutions of (local/global) Dirichlet problems.	Element
`RCLoop {<rcloop-entries>}`	Specification of tree loop properties.	–
`[-] Poisson`	Enables/disables EPC smoothing step.	enabled
`[-] Smooth`	Enables/disables NBJI smoothing step.	enabled

Device parameters affecting smoothing

The flags `[-]Poisson` and `[-]Smooth` enable or disable the EPC smoothing step and the NBJI, respectively.

The avalanche homotopy can be influenced only by a temporary interface via environment variables. The environment variable `DES_AGM_AVAHOMOTOPY_NB_IT` modifies the number of iterations used within Newton iterations of the homotopy, for example, for a C-shell, the command:

```
setenv DES_AGM_AVAHOMOTOPY_NB_IT 10
```

sets the number of iterations (which is internally multiplied by 5). Setting `DES_AGM_AVAHOMOTOPY_NB_IT` to 0 disables the avalanche homotopy (default is 5). The environment variable `DES_AGM_AVAHOMOTOPY_LINPAR` causes a linear parameter ramping in the homotopy, and `DES_AGM_AVAHOMOTOPY_EXTRAPOL` switches the extrapolation on within the homotopy.

31.2.2 Grid specification

Adaptive device instances require a grid description in the form of MESH command and boundary files. This is specified by the keyword `Boundary` in the `File` section of the instance. If `Boundary` is specified, the keywords `Grid` and `Doping` of the `File` section have a different interpretation: These strings now serve as the basis for output file names of grid and doping information (required to visualize corresponding plot files).

An example `File` section is:

```
File {
   Boundary  = "mos_agm_msh"      # INPUT files "mos_agm_msh.{cmd,bnd}"
   Grid      = "test_agm_des"     # reinterpreted as OUTPUT !
   Doping    = "test_agm_des"     # reinterpreted as OUTPUT !
   ...
   GridCompressed                 # writing compressed grid and doping files
}
```

With the keyword `GridCompressed`, the grid and doping files are written as compressed files (`.gz`).

The specification in the MESH command file is used to construct the initial mesh for the device. During adaptation, the maximal edge length specifications are respected, that is, in this way it is possible to control the coarsest possible grid. The minimal edge length specifications describe the finest possible mesh during adaptation, that is, elements are not further refined if half of the edge length is smaller than the specified minimal value. The doping-specific refinement information is completely ignored during adaptation, that is, it is only relevant for initial grid generation.

DESSIS supports grid adaptation for the quadtree grid generation approach of MESH, though the grid generation algorithm used in DESSIS differs slightly from the algorithm of the (stand-alone) mesh generator MESH:

- Minimal edge length specifications of refinement definitions are interpreted individually per referencing refinement placement (instead of using the smallest, local, minimal edge length value of all refinement definitions).
- The local lower bound of edge lengths allowed in AGM is the smallest, local, minimal edge length of all referenced refinement definitions.
- The `NOFFSET` section in the command file causes an error during parsing and should be removed.

NOTE DESSIS supports the MESH command file syntax of MESH Release 7.0.

31.3 Adaptation criteria

The adaptation criteria are specified as AGM device body entries and determine if adaptation (refinement and/ or coarsening) is required, for example:

```
GridAdaptation ( ... ) {
   Criteria {            # list of adaptation criteria
      Dirichlet (DataName = "DomainIntegralCurrent" Contact = "drain"
                 AbsError = 1.e-7 RelError = 0.2)
      Residual (DataName = "AGMDissipationRate"
                  AbsError = 1.e-20 RelError = 0.4)
         Element (DataName = "ElectrostaticPotential"
               MaxTransDiff = 0.1)
      }
   }
```

31.3.1 General

Each adaptation criterion refers to a physical quantity specified by `DataName = "<quantity-name>"`. The supported data for the different types of criterion are listed in Table 15.160.

Table 15.160 Supported data for different adaptation types of criterion

Criterion type	**Supported data**
Dirichlet	`AGMDissipationRate`, `DomainIntegralCurrent`
Residual	`AGMDissipationRate`
Element	Any vertex-based scalar datasets known in DESSIS.

The relative and absolute error tolerances ε_R and ε_A are specified using `RelError=<real>` or `AbsError=<real>` (not used for element criteria). Each adaptation criterion plots specific data to plot files in addition to the specified data of the `Plot` section.

31.3.2 Dirichlet

For the functional `DomainIntegralCurrent`, an additional contact name must be supplied to the criterion using the keyword `Contact`. The flag `CurrentWeighting` must be switched on in the `Math` section of the device. Dirichlet-type criteria plot the specified data, and its error and deviation.

NOTE With the current implementation of the element patch mode, Dirichlet error estimation is very time-consuming and, therefore, such error indicators should be used only if they improve the grid adaptation compared to other criteria.

31.3.3 Residual

No additional options are available. Residual-type criteria plot the specified data and its error.

31.3.4 Element

The element is refined if $\eta_T(f) > d_{mtd}$ where d_{mtd} is the value specified by `MaxTransDiff = <real>`, for example:

```
   Criteria {
      Element (DataName = "ElectrostaticPotential" # vertex based data
               MaxTransDiff = 0.1                  # unit is taken from DATEX
      }
   }
```

All data that can be plotted is available as a vertex-based dataset and can be used in the element-type criterion. The specified data is plotted by this criterion type.

31.4 Adaptive solve statements

Grid adaptation is only performed for explicitly adaptive solve statements using the keyword `GridAdaptation`. Only the highest level `Coupled` and `Quasistationary` solve statements can be adaptive.

31.4.1 General adaptive solve statements

The following parameter entries are interpreted by all adaptive solve statements.

With `MaxCLoops`, the maximal number of adaptation iterations for each adaptive coupled system is given (default is 100000). The flag `[-]Plot` enables or disables device plots for all intermediate device grids (default is disabled). The flag `[-]CurrentPlot` allows for the plotting of current file data on intermediate grids to the current file (default is disabled).

31.4.2 Adaptive coupled solve statements

A `Coupled` solve statement can be adaptive if (a) it is the highest level solve statement and (b) it contains either none or at least the three drift-diffusion equations (`Poisson`, `Electron`, and `Hole`) for all adaptive devices:

```
Coupled ( ... GridAdaptation ( MaxCLoops = 5 ) )
      { Poisson Electron Hole }
```

31.4.3 Adaptive quasistationary solve statements

In the current implementation, a `Quasistationary` can be adaptive only if (a) it is the highest level solve statement and (b) its system consists of a `Coupled` solve statement (containing none or, at least, the `Poisson`, `Electron`, and `Hole` equations of adaptive devices), that is, `Plugin` statements are not yet supported.

In adaptive quasistationary simulations, it may be useful to restrict the adaptation to certain ranges of the parameter values. This can be achieved by specifying parameter ranges or iteration numbers in a similar fashion as in `Plot` in solve statements using the `Time`, `IterationStep`, and `Iterations` keywords, for example:

```
Quasistationary (
   GridAdaptation (
      IterationStep = 10
      Iterations = ( 2 ; 7 )
      Time = ( Range = ( 0.2 0.4 ) ; range = (0. 1.) intervals = 5 ; 0.1 ; 0.99 )
      ...
   ) ...
) { ... }
```

The fixed times specified by `Time` do not force adaptation at these values (which can be achieved by other methods, for example, adding plot statements for the desired parameter values), while the free time ranges (specified by `Range` without the keyword `Intervals`) force adaptation if the actual parameter falls into the specified open interval.

31.5 Limitations and recommendations

31.5.1 Limitations

The grid adaptation approach implemented in DESSIS has been developed for the drift-diffusion model and 2D quadtree-based simulation grids, and is still under development. For the convenience of the user, the approach has been formally extended to support other transport models and features available in DESSIS, that is, AGM is formally compatible with the drift-diffusion, thermodynamic, and hydrodynamic transport models. Nevertheless, it must be noted that AGM has been applied so far only for the drift-diffusion model and silicon devices. Total incompatibility is to be expected with the Schrödinger equation solver, heterostructures, interface conditions, and laser equations. These incompatibilities are only partially checked after command file parsing.

31.5.2 Recommendations

Accuracy of terminal currents as adaptation goal

The choice of appropriate adaptation criteria depends on the adaptation goal. In most adaptive simulation procedures, the local discretization errors are used as adaptation criteria. This approach is not practicable for device simulation as the criteria lead to overwhelmingly large grid sizes. The residual and Dirichlet adaptation criteria for the functionals `AGMDissipationRate` and `DomainIntegralCurrent` aim for accurate computations of the device terminal currents, a minimal requirement for all simulation cases, allowing in principle some unresolved solution layers, which do not contribute to the terminal current computation.

AGM simulation times

Each coupled adaptation iteration requires remarkable simulation time. In contrast to linear or easy-to-solve problems, for device simulation, most time is consumed in the recomputation procedure of the solution due to the extreme nonlinearities of the problem and not in the pure adaptation of the grid. The Dirichlet adaptation criteria require the solution of local or global nonlinear problems that also consume large parts of the simulation time even if the grid is not to be updated. Before using very time-consuming adaptation criteria, it is recommended to use first explicit adaptation criteria (not requiring the solution of additional equations, such as residual or element variation criteria) to see which kinds of mesh are created and if the AGM module runs robustly on the given simulation.

As a second step, it may be useful to add Dirichlet adaptation criteria as they provide more accurate (mathematical) error bounds than the other error indicators. The most time-consuming parts in many AGM simulations are (in order of importance):

1. Dirichlet adaptation criteria with Element path mode computations (very expensive).
2. Avalanche homotopy if the computation for $t = 1$ fails to converge (can be very expensive).
3. Dirichlet adaptation criteria with Global patch mode computations.
4. NBJI smoothing step (for large grid sizes).
5. EPC smoothing step.
6. Grid generation.

Dirichlet versus residual adaptation criteria

The Dirichlet adaptation criteria are implicit adaptation criteria, that is, they require solutions of local boundary value problems, while the residual adaptation criteria are explicit, that is, refer only to the solution on the actual grid. Therefore, the Dirichlet criteria are much more expensive. Experimentally, the Dirichlet and residual criteria for `AGMDissipationRate` are comparable in wide regions of the device, that is:

$$\eta_F^{\text{Residual}}(D_{AGM}) \approx c\eta_F^{\text{Dirichlet}}(D_{AGM}) \tag{15.626}$$

where $c \approx 4$ for (essentially) 1D and $c \approx 2$ for 2D simulations. This relationship makes the residual error indicator a very favorable choice as it is much cheaper and, in general, smoother than the Dirichlet error indicator.

Large grid sizes

The low convergence order of the discretization causes quite large grid sizes even for low accuracy requirements. Especially in the vicinity of solution layers and singularities (at boundary points with changing boundary conditions), the point density is hard to control. The user has several possibilities to influence the sizes of the resulting grids:

- Increase minimal edge length specification in the MESH command file: Elements that reach the allowed minimal edge length are no longer refined and their local error does not contribute to the global error used in the adaptation decision. Hence, realistic minimal edge lengths stop refinement in singularities and layers.
- Decrease the accuracy requirement in adaptation criteria. Realistic relative error tolerances for the Dirichlet adaptation criteria are about 0.1 or greater. The comparison relation between Dirichlet and residual adaptation criteria results in relative error tolerances `RelError > 0.4` for the residual criteria.
- Reduce the maximal number of elements in the macro element tree (`MaxNumberMacroElements`).

Convergence problems after adaptation

It has been observed that, for very coarse simulation grids, the recomputation procedure shows convergence problems. Such problems can be solved by using slightly refined initial grids or by refining the coarsest possible grid (reduce `MaxElementSize` in refinement definitions). On the other hand, the changes between consecutive grids could be too large, which can be controlled by reducing the number of tree loop iterations. Convergence problems occur in the presence of strong nonlinearities (caused by the selected transport model or certain physical models). Restricting the physical complexity may solve the problem and resulting grids may be still useful for the intended simulation case.

AGM and extrapolate

After adaptation within a quasistationary, extrapolation is not possible and the parameter step size may decrease. Extrapolation is supported as soon as two consecutive solutions are computed on the same mesh.

CHAPTER 32 Numeric methods

32.1 Discretization

The well-known ‘box discretization’ [1][99][100] is applied to discretize the partial differential equations (PDEs). This method integrates the PDEs over a test volume such as that shown in Figure 15.115, which applies the Gaussian theorem, and discretizes the resulting terms to a first-order approximation.

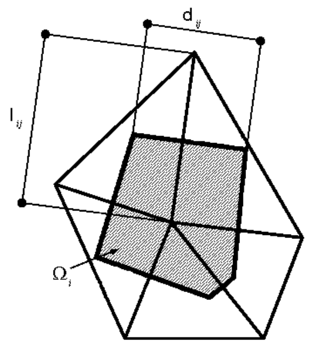


Figure 15.115 Single box for a triangular mesh in 2D

In general, box discretization discretizes each PDE of the form:

$$\nabla \cdot \vec{J} + R = 0 \tag{15.627}$$

into:

$$\sum_{j \neq i} \kappa_{ij} \cdot j_{ij} + \mu(\Omega_i) \cdot r_i = 0 \tag{15.628}$$

with values listed in Table 15.161.

Table 15.161 Dimension

Dimension	κ_{ij}	$\mu(\Omega_i)$
1D	$1 / l_{ij}$	Box length
2D	d_{ij} / l_{ij}	Box area
3D	D_{ij} / l_{ij}	Box volume

In this case, the physical parameters j_{ij} and r_i have the values listed in Table 15.162, where $B(x) = x/(e^x - 1)$ is the Bernoulli function.

Table 15.162 Equation

Equation	j_{ij}	r_i
Poisson	$\varepsilon(u_i - u_j)$	$-\rho_i$
Electron continuity	$\mu^n(n_i B(u_i - u_j) - n_j B(u_j - u_i))$	$R_i - G_i + \frac{d}{dt} n_i$
Hole continuity	$\mu^p(p_j B(u_j - u_i) - p_i B(u_i - u_j))$	$R_i - G_i + \frac{d}{dt} p_i$
Temperature	$\kappa(T_i - T_j)$	$H_i - \frac{d}{dt} T_i c_i$

One special feature of DESSIS is that the actual assembly of the nonlinear equations is performed elementwise, that is:

$$\sum_{e \in Elements(i)} \left\{ \left(\sum_{j \in Vertices(e)} \kappa_{ij}^e \cdot j_{ij}^e \right) + \mu^e(\Omega_i) \cdot r_i^e \right\} = 0 \tag{15.629}$$

This expression is equivalent to (Eq. 15.628), but has the advantage that some parameters (such as ε, μ_n, μ_p) can be handled elementwise, which is useful for numeric stability and physical exactness.

32.2 Box method coefficients

32.2.1 Basic definitions

A mesh is a Delaunay mesh if the interior of the circumsphere (circumcircle for 2D) of each element contains no mesh vertices.

Let T be a mesh element. The center circumsphere (circle for 2D) around the element T is called the element Voronoï center V_T. Let f be the face of the element T. The center circumcircle around the face f is called the face Voronoï center V_f.

Let v be a vertex of the mesh and let $ev^n (1 \le n \le N)$ be the set of edges connected to vertex v. Let P_{ev}^n be the mid-perpendicular plane for the edge ev^n. The plane P_{ev}^n splits 3D space into two half-spaces. Let S_{ev}^n be the half-space that contains the vertex v. The intersection of all half-spaces S_{ev}^n is called the Voronoï box B_v of vertex v. The Voronoï box B_v is the convex polyhedron and any face of B_v is called a Voronoï face. In addition, $T_v^m (1 \le m \le M)$ is the set of elements per vertex v and $T_{ev}^k (1 \le k \le K)$ is the set of elements per edge ev.

For a Delaunay mesh, there are two propositions:

1. The vertices of a Voronoï box B_v are element Voronoï centers, that is, B_v is a polyhedron with the vertices $(V_{T_v^1}, V_{T_v^2}, \ldots, V_{T_v^M})$.
2. The Voronoï face associated with the edge ev is the convex polygon with the vertices $(V_{T_{ev}^1}, V_{T_{ev}^2}, \ldots, V_{T_{ev}^K})$ and this polygon lies in the mid-perpendicular plane P_{ev} (see Figure 15.116).

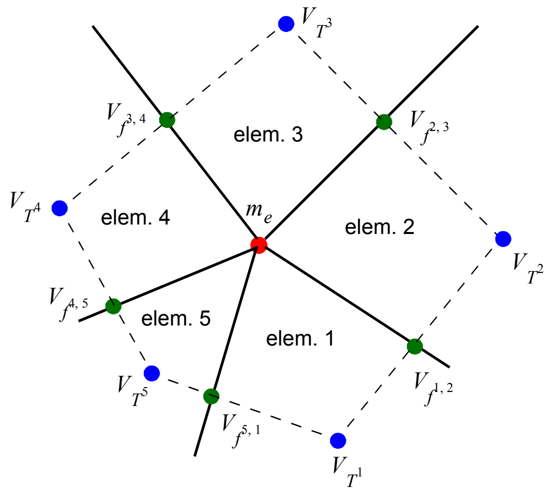


Figure 15.116 Voronoï face of 3D Delaunay elements: view of mid-perpendicular plane at edge e with element Voronoï centers V_{T^i} and the face Voronoï center $V_{f^{i,i+1}}$ between elements T^i and T^{i+1}

For a non-Delaunay mesh, the Voronoï box is a convex polyhedron, but there are vertices that are not Voronoï element centers (see Figure 15.117, vertex R).

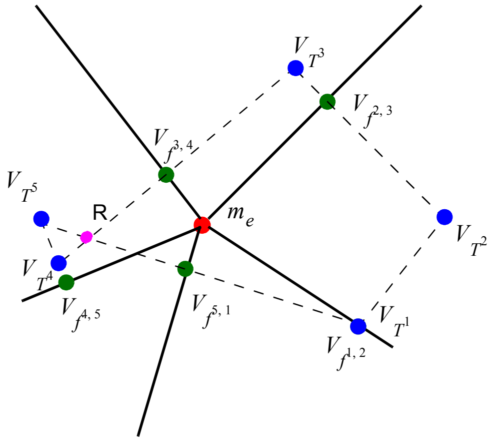


Figure 15.117 Voronoï face of 3D non-Delaunay elements: non-Delaunay mesh; Voronoï face is polygon $(V_{T^1}, V_{T^2}, V_{T^3}, R, V_{T^1})$

32.2.2 Variation of box method algorithms

The various box methods differ in the way that the 2D volume of the Voronoï face is split into the elements T_{ev}^k associated with edge ev. All box methods give the same result if the mesh has no obtuse elements (an element is obtuse if its element Voronoï center is outside of the element).

DESSIS implements of the following box method (BM) algorithms: element intersection BM, element-face intersection BM, and quadrilateral BM. Figure 15.118 shows a situation where these methods produce different results. For all methods, the Voronoï faces for elements T^1, T^2, T^3 are the same and are equal to the area of the polygons $(m_e, V_{f^{i-1,i}}, V_{T^i}, V_{f^{i,i+1}}, m_e)$. The elements T^4, T^5 have the following Voronoï faces depending on the BM algorithms:

- Element intersection BM algorithms
  - Element T^4 : area of polygon $(m_e, V_{f^{3,4}}, V_{T^4}, V_{T^5}, p, m_e)$
  - Element T^5 : area of polygon $(m_e, p, V_{f^{5,1}}, m_e)$
- Element-face intersection BM algorithms
  - Element T^4 : area of polygon $(m_e, V_{f^{3,4}}, V_{T^4}, V_{T^5}, m_e)$
  - Element T^5 : area of polygon $(m_e, V_{T^5} V_{f^{5,1}}, m_e)$
- Quadrilateral BM algorithms
  - Element T^4 : area of polygon $(m_e, V_{f^{3,4}}, V_{T^4}, V_{f^{4,5}}, m_e)$
  - Element T^5 : area of polygon $(m_e, V_{T^5} V_{f^{5,1}}, m_e)$ minus area of polygon $(m_e, V_{T^5} V_{f^{4,5}}, m_e)$

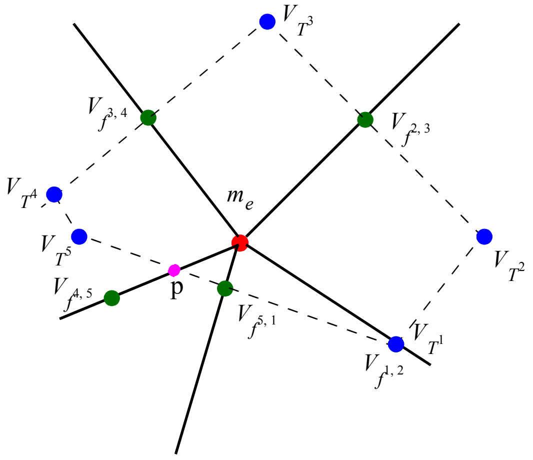


Figure 15.118 Voronoï face of 3D elements to consider different box method algorithms

32.2.3 Truncated and non-Delaunay elements

If an obtuse element has an obtuse face on the material or region boundary, then for this element, an algorithm of truncation can be applied. Figure 15.119 on page 15.523 shows the difference between the original and truncated Voronoï polygons in the 2D case.

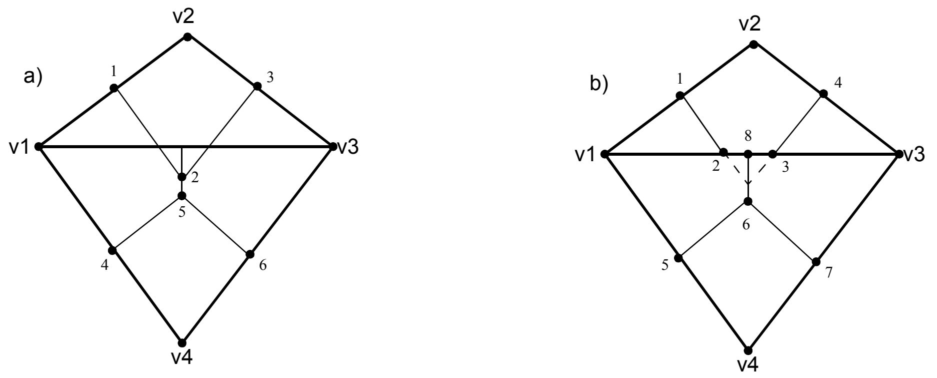


Figure 15.119 Box method in 2D for truncated element: (*a*) Voronoï polygons before truncation – P1(v1,1,2,5,4,v1), P2(v2,1,2,3,v2), P3(v3,3,2,5,6,v3), P4(v4,4,5,6,v4); (*b*) Voronoï polygons after truncation – P1(v1,1,2,8,6,5,v1), P2(v2,1,2,3,4,v2), P3(v3,4,3,8,6,7,v3), P4(v4,5,6,7,v4)

For the 3D case, a similar algorithm of truncation is used. If an element is non-Delaunay, it is truncated in any case. In the 2D case, there is a soft truncation for the non-Delaunay element if the neighbor elements have the same material (see Figure 15.120).

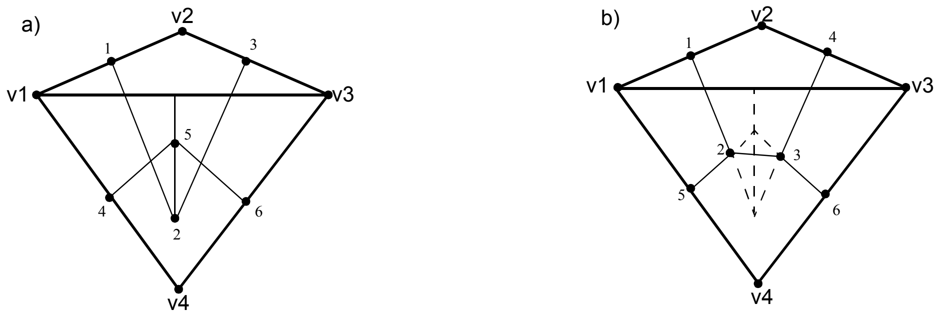


Figure 15.120 Box method in 2D for non-Delaunay element: (*a*) Voronoï polygons before truncation – P1(v1,1,2,5,4,v1), P2(v2,1,2,3,v2), P3(v3,3,2,5,6,v3), P4(v4,4,5,6,v4) (the polygons P1 and P3 are singular; the polygons P2 and P4 have an intersection); (*b*) Voronoï polygons after truncation – P1(v1,1,2,5,v1), P2(v2,1,2,3,4,v2), P3(v3,4,3,6,v3), P4(v4,5,2,3,6,v4) (the polygons P1 and P3 are not singular; the polygons P2 and P4 have no intersection)

32.2.4 Math parameters for box method coefficients

The coefficients needed for discretization $\mu^e(\Omega_i)$ and κ^e_{ij} from (Eq. 15.629) (referred to as `Measure` and `Coefficients`) can be computed inside DESSIS or read from outside files.

Table 15.163 on page 15.524 lists all available options for computing `Measure` and `Coefficients`.

Table 15.163 Keywords for box method coefficients

Keyword	Description
`-AverageBoxMethod`	Quadrilateral BM algorithm.
`AverageBoxMethod`	Element intersection BM algorithm that is element oriented, that is, for each element and all edges of this element, it computes the element Voronoï faces.
`NaturalBoxMethod`	Element intersection BM algorithm that is edge oriented, that is, for each edge and all elements around this edge, it computes the element Voronoï faces.
`VoronoiFaceBoxMethod = -TruncatedVoronoiBox`	Element-face intersection BM algorithm without truncated correction for obtuse elements. For non-Delaunay elements, it uses truncated correction in any case.
`VoronoiFaceBoxMethod = TruncatedVoronoiBox`	Element-face intersection BM algorithm with truncated correction for all obtuse elements.
`VoronoiFaceBoxMethod = RegionBoundaryVoronoiBox`	Element-face intersection BM algorithm with truncated correction for obtuse elements, which have an obtuse face (edge for 2D) on boundary of regions, or these elements are not Delaunay.
`VoronoiFaceBoxMethod = MaterialBoundaryVoronoiBox`	Element-face intersection BM algorithm with truncated correction for obtuse elements, which have an obtuse face (edge for 2D) on boundary of materials, or these elements are not Delaunay.
`[-]BoxMethodFromFile`	Read Voronoï surface from grid file. By default, this option is switched on. If the grid file has a `VoronoiFaces` section, all the above options are ignored and the Voronoï surface is read from this file. A `VoronoiFaces` section is added to the grid file if MESH is called with the option `-voronoiOutput` (used in quadrilateral BM algorithm).

`[-]BoxCoefficientsFromFile`
`[-]BoxMeasureFromFile`
	Both keywords try to read sections of the geometry file (see Section 32.2.5).

Only one BM algorithm can be activated. The default options are:

```
Math { ...
    AverageBoxMethod  -VoronoiFaceBoxMethod    -NaturalBoxMethod
    BoxMethodFromFile -BoxCoefficientsFromFile -BoxMeasureFromFile
    ...
}
```

32.2.5 Saving and restoring box method coefficients

Usually, the coefficients needed for discretization are computed inside DESSIS. For experimental purposes, it may be preferred to use externally provided data. `Measure` and `Coefficients` can be stored in an input geometry file (it is an old format for input mesh file) or in the file `MeasureCoefficientsDebug`. DESSIS reads them from the file if the keyword `BoxMeasureFromFile` (to read the `Measure` array) or `BoxCoefficientsFromFile` (to read the `Coefficients` array) is specified in the `Math` section of the DESSIS input file.

`Measure[element][element_vertex_index]` is the control volume associated with the index `element_vertex_index` of the element `element` ($\mu^e(\Omega_i)$ from (Eq. 15.629)). In line `k` of the `Measure` section, the control volume for each `element_vertex_index` of element `k` is stored. The `Measure` section in the grid file appears as:

```
Measure {
   8.719666833501378e-08 4.359833416750702e-08 4.359833416750729e-08
   8.719666833501378e-08 4.359833416750702e-08 4.359833416750729e-08
   ...
}
```

`Coefficient[element][element_edge_index]` is the ratio of the cross section of the appropriate control volume of an element to the length of the edge corresponding to `element_edge_index` (κ_{ij}^e from (Eq. 15.629)). In the line `k` of the `Coefficients` section, the ratios for each `element_edge_index` of element `k` are stored.

Usually, it is not necessary to write `Measure` and `Coefficients`, although for debugging purposes, it may be useful. DESSIS writes this information into the file `MeasureCoefficientsDebug` if the keyword `BoxMeasureFromFile` or `BoxCoefficientsFromFile` is specified.

32.3 AC simulation

AC simulation is based on small-signal AC analysis. The response of the device to ‘small’ sinusoidal signals superimposed upon an established DC bias is computed as a function of frequency and DC operating point. Steady-state solution is used to build up a linear algebraic system [196] whose solution provides the real and imaginary parts of the variation of the solution vector $(\psi, n, p, T_n, T_p, T_L)$ induced by small sinusoidal perturbation at the contacts.

32.3.1 AC response

The AC response is obtained from the three basic semiconductor equations (see (Eq. 15.19) and (Eq. 15.20)) and from up to three additional energy conservation equations to account for electron, hole, and lattice temperature responses. In the following description of the AC system, the temperatures have been omitted in the solution vector and Jacobian for simplicity, a complete description being formally obtained by adding the temperature responses to the solution vector and the corresponding lines to the system Jacobian. After discretization, the simplified system of equations can be symbolically represented at the node i of the computation mesh as:

$$F_{\psi i}(\psi, n, p) = 0 \tag{15.630}$$

$$F_{ni}(\psi, n, p) = \dot{G}_{ni}(n) \tag{15.631}$$

$$F_{pi}(\psi, n, p) = \dot{G}_{pi}(p) \tag{15.632}$$

where F and G are nonlinear functions of the vector arguments ψ, n, p, and the dot denotes time differentiation.

By substituting the vector functions of the form $\xi_{total} = \xi_{DC} + \tilde{\xi} e^{j\omega t}$ into (Eq. 15.630), (Eq. 15.631), and (Eq. 15.632) where $\xi = \psi, n, p$, ξ_{DC} is the value of ξ at the DC operating point, and $\tilde{\xi}$ is the corresponding response (or the phasor uniquely identifying the complex perturbation) and then expanding the nonlinear functions F and G in the Taylor's series around the DC operating point and keeping only the first-order terms (the small-signal approximation), the AC system of equations at the node i can be written as:

$$\sum_j \begin{bmatrix} \frac{\partial F_{\psi i}}{\partial \psi_j} & \frac{\partial F_{\psi i}}{\partial n_j} & \frac{\partial F_{\psi i}}{\partial p_j} \\ \frac{\partial F_{ni}}{\partial \psi_j} & \frac{\partial F_{ni}}{\partial n_j} - j\omega \frac{\partial G_{ni}}{\partial n_j} & \frac{\partial F_{ni}}{\partial p_j} \\ \frac{\partial F_{pi}}{\partial \psi_j} & \frac{\partial F_{pi}}{\partial n_j} & \frac{\partial F_{pi}}{\partial p_j} - j\omega \frac{\partial G_{pi}}{\partial p_j} \end{bmatrix}_{DC} \begin{bmatrix} \tilde{\psi}_j \\ \tilde{n}_j \\ \tilde{p}_j \end{bmatrix} = 0 \tag{15.633}$$

where the solution vector is scaled with respect to terminal voltages (at the contact where the voltage is applied, $\tilde{\psi}$ is 1). Therefore, the unit of carrier density responses is cm^{-3} V^{-1} and the potential response is unitless.

The matrix of (Eq. 15.633) differs from the Jacobian of the system of equations (Eq. 15.630), (Eq. 15.631), and (Eq. 15.632) only by pure imaginary additive terms involving derivatives of G with respect to carrier densities. The global AC matrix system is obtained by imposing the corresponding AC boundary conditions and performing the summation (assembling the global matrix).

Common AC boundary conditions used in AC simulation are Neumann boundary and oxide–semiconductor jump conditions carried over directly from DC simulation; Dirichlet boundary conditions for carrier densities where n and p at Ohmic contacts are $\tilde{n} = \tilde{p} = 0$; and Dirichlet boundary conditions for AC potential at Ohmic contacts that are used to excite the system.

After assembling the global AC matrix and taking into account the boundary conditions, the AC system becomes:

$$[J + jD]\tilde{X} = B \tag{15.634}$$

where J is the Jacobian matrix, D contains the contributions of the G functions to the matrix, B is a real vector dependent on the AC voltage drive, and $\tilde{X}$ is the AC solution vector. By writing the solution vector as $\tilde{X} = X_R + jX_I$ with X_R and X_I the real and imaginary part of the solution vector respectively, the AC system can be rewritten using only real arithmetic as:

$$\begin{bmatrix} J & -D \\ D & J \end{bmatrix} \begin{bmatrix} X_R \\ X_I \end{bmatrix} = \begin{bmatrix} B \\ 0 \end{bmatrix} \tag{15.635}$$

The AC response is actually computed by solving the real system (Eq. 15.635).

An `ACPlot` statement in the `System` section is used to plot AC responses $(\tilde{\psi}, \tilde{n}, \tilde{p}, \tilde{T}_n, \tilde{T}_p, \tilde{T}_L)$. The responses are plotted in the DESSIS AC plot file with a separate file for each frequency.

For details of the `ACPlot` statement, see Table 15.50 on page 15.117. For details and examples of small-signal AC analysis, see Section 3.8.3 on page 15.117.

32.3.2 AC current density responses

When the AC system is solved, the AC current density responses $\overrightarrow{\tilde{J}_{disp}}$, $\vec{\tilde{J}_n}$, and $\vec{\tilde{J}_p}$ are computed using:

$$\overrightarrow{\tilde{J}_{disp}} = -j\omega\varepsilon\nabla\tilde{\psi} \tag{15.636}$$

$$\vec{\tilde{J}_n} = \left.\frac{\partial\vec{J}_n}{\partial\psi}\right|_{DC}\tilde{\psi} + \left.\frac{\partial\vec{J}_n}{\partial n}\right|_{DC}\tilde{n} + \left.\frac{\partial\vec{J}_n}{\partial p}\right|_{DC}\tilde{p} \tag{15.637}$$

$$\vec{\tilde{J}_p} = \left.\frac{\partial\vec{J}_p}{\partial\psi}\right|_{DC}\tilde{\psi} + \left.\frac{\partial\vec{J}_p}{\partial p}\right|_{DC}\tilde{p} + \left.\frac{\partial\vec{J}_p}{\partial n}\right|_{DC}\tilde{n} \tag{15.638}$$

The unit of current density responses is Acm^{-2} V^{-1}.

The `ACPlot` statement in the `System` section is used to plot the six AC current density responses.The responses are added to the AC solution response in the DESSIS AC plot files.

32.4 Transient simulation

Transient equations used in semiconductor device models and circuit analysis can be formally written as a set of ordinary differential equations:

$$\frac{d}{dt}q(z(t)) + f(t, z(t)) = 0 \tag{15.639}$$

which can be mapped to the DC and transient parts of the PDEs. DESSIS uses implicit discretization of transient equations (see (Eq. 15.639)), and supports two discretization schemes: simple backward Euler (BE), and composite trapezoidal rule/backward differentiation formula (TRBDF), which is the default.

32.4.1 Backward Euler method

Backward Euler is a very stable method, but it has only a first-order of approximation over time-step h$_n$. The discretization can be written as:

$$q(t_n + h_n) + h_n f(t_n + h_n) = q(t_n) \tag{15.640}$$

The local truncation error (LTE) estimation is based on the comparison of the obtained solution $q(t_n{+}h_n)$ with the linear extrapolation from the previous time-step. The extrapolated solution is written as:

$$q^{extr} = q(t_n) - \frac{f(t_n) + f(t_n + h_n)}{2}h_n \tag{15.641}$$

Then, in every point, the relative error can be estimated as $((q(t_n + h_n)) - q^{extr})/(q(t_n + h_n))$.

Using (Eq. 15.640) and (Eq. 15.641), and estimating the norm of relative error, DESSIS computes the value:

$$r = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{f(t_n+h_n)-f(t_n)}{\varepsilon_R|q_n(t_n+h_n)|+\varepsilon_A h_n}\right)^2} \tag{15.642}$$

where the sum is taken over all unknowns (that is, all free vertices of all equations), and $\varepsilon_{R,tr}$ and $\varepsilon_{A,tr}$ are the relative and absolute transient errors, respectively.

The next time-step is estimated as:

$$h_{est} = h_n r^{-1/2} \tag{15.643}$$

The value of the estimated time-step is used for h_{n+1} computation (see Section 32.4.3 on page 15.529).

32.4.2 TRBDF composite method

The transient scheme [101] for the approximation of (Eq. 15.9) is briefly reviewed in this section. From each time point t_n, the next time point $t_n + h_n$ (h_n is the current step size) is not directly reached. Instead, a step in between to $t_n + \gamma h_n$ is made. This improves the accuracy of the method. $\gamma = 2 - \sqrt{2}$ has been shown to be the optimal value. Using this, two nonlinear systems are reached. For the trapezoidal rule (TR) step:

$$2q(t_n+\gamma h_n)+\gamma h_n f(t_n+\gamma h_n) = 2q(t_n)-\gamma h_n f(t_n) \tag{15.644}$$

and for the BDF2 step:

$$(2-\gamma)q(t_n+h_n)+(1-\gamma)h_n f(t_n+h_n) = (1/\gamma)(q(t_n+\gamma h_n)-(1-\gamma)^2 q(t_n)) \tag{15.645}$$

The local truncation error (LTE) is estimated after such a double step as:

$$\tau = \left[\frac{f(t_n)}{\gamma} - \frac{f(t_n+\gamma h_n)}{\gamma(1-\gamma)} + \frac{f(t_n+h_n)}{1-\gamma}\right] \tag{15.646}$$

$$C = \frac{-3\gamma^2+4\gamma-2}{12(2-\gamma)} \tag{15.647}$$

DESSIS then computes the following value from this:

$$r = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\tau_i}{\varepsilon_R|q_n(t_n+h_n)|+\varepsilon_A}\right)^2} \tag{15.648}$$

where the sum is taken over all unknowns (that is, all free vertices of all equations), and $\varepsilon_{R,tr}$ and $\varepsilon_{A,tr}$ are the relative and absolute transient errors, respectively. Since the TRBDF method has a second-order approximation over h_n, the next step can be estimated as:

$$h_{est} = h_n r^{-1/3} \tag{15.649}$$

The value of the estimated time-step is used for h_{n+1} computation (see Section 32.4.3).

32.4.3 Syntax and implementation

By default, DESSIS uses the TRBDF method. To switch to backward Euler (BE), the statement `Transient=BE` must be specified in the `Math` section.

To evaluate whether a time-step was successful and to provide an estimate for the next step size, the following rules are applied:

- If one of the nonlinear systems cannot be solved, the step is refused and tried again with $h_n = 0.5 \cdot h_n$.
- Otherwise, the inequality $r < 2f_{rej}$ is tested. If it is fulfilled, the transient simulation proceeds with $h_{n+1} = h_{est}$. Otherwise, the step is re-tried with $h_n = 0.9 \cdot h_{est}$.
- The LTE is checked only if the `CheckErrorTransient` option is selected; otherwise, the selection of the next time-step is based only on convergence of nonlinear iterations.

To activate LTE evaluation and time-step control, `CheckErrorTransient` must be specified either globally (in the `Math` section) or locally as an option in the `Transient` statement. The keyword `NoCheckErrorTransient` disables time-step control. The value of the relative error is defined by the parameter `TransientDigits` according to (Eq. 15.15).

Absolute error is given by the keyword `TransientError` or recomputed from `TransientErrRef` ($x_{\mathrm{ref,tr}}$) using (Eq. 15.16) (if `RelErrControl` is switched on). DESSIS provides the default values of $\varepsilon_{R,tr}$, $\varepsilon_{A,tr}$, and $x_{\mathrm{ref,tr}}$. The coefficient f_{rej} is equal to 1 by default. The user can define the values of $\varepsilon_{R,tr}$, $\varepsilon_{A,tr}$, $x_{\mathrm{ref,tr}}$, and f_{rej} globally in the `Math` section, or specify them as options in the `Transient` statement. In the latter case, it overwrites the default and `Math` specifications for this command.

32.5 Nonlinear solvers

In the next two sections, the *Digits* variable corresponds to the keyword `Digits`, which can be given in the `Math` section of the input file (see Section 2.10.2 on page 15.76), or in parentheses of each `Plugin` or `Coupled` statement.

32.5.1 Full coupled solution

For the solution of nonlinear systems, the scheme developed by Bank and Rose [102] is applied. This scheme tries to solve the nonlinear system $g(z) = 0$ by the Newton method:

$$\vec{g} + \vec{g}'\vec{x} = 0 \tag{15.650}$$

$$\vec{z}^{j} - \vec{z}^{j+1} = \lambda\vec{x} \tag{15.651}$$

where λ is selected such that $\|g_{k+1}\| < \|g_k\|$, but is as close as possible to 1. DESSIS handles the error by computing an error function that can be defined by two methods.

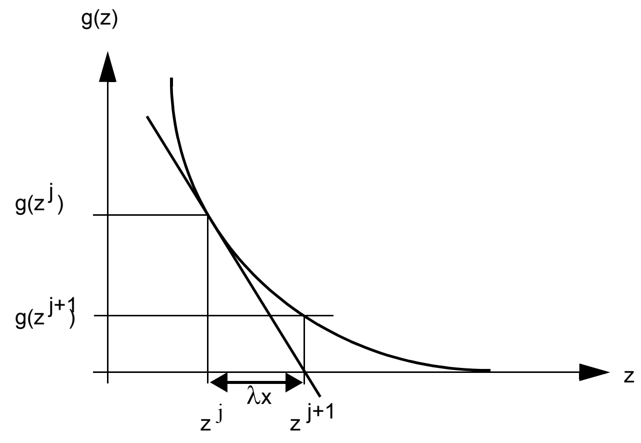


Figure 15.121 Newton iteration

The Newton iterations stop if the convergence criteria are fulfilled. One convergence criterion is the norm of the right-hand side, that is, $\|g\|$ in (Eq. 15.650). Another natural criterion may be the relative error of the variables measured, such as $\left\|\frac{(\lambda x)}{z}\right\|$.

Conversely, for the very small z updates, λx must be measured with respect to some reference value of the variable z_{ref}. The formula used in DESSIS as the second convergence criterion is:

$$\frac{1}{\varepsilon_R}\frac{1}{\mathrm{Norm}}\sum_{i,EQ}\frac{|z(\mathrm{EQ,i,j}) - z(\mathrm{EQ,i,j\text{-}1})|}{|z(\mathrm{EQ,i,j})| + z_{ref}(\mathrm{EQ})} < 1 \tag{15.652}$$

where $z(\mathrm{EQ,i,j})$ is the solution of the equation *EQ* (*EQ* is *Poisson, Electron, Hole*, and so on) at the node *i* after the Newton iteration of *j*. The constant *Norm* is given by the total number of nodes multiplied by the total number of equations. The parameter ε_R is the relative error criterion. The value of $\varepsilon_R = 10^{\text{-Digits}}$ is set by specifying the following in the `Math` section:

```
Math{ ...
   Digits = 5
}
```

where 5 is the default for `Digits`. The reference values $z_{ref}(\mathrm{EQ})$ ensure numeric stability even for cases when $z(\mathrm{EQ,i,j})$ is zero or very small. This error condition ensures that the respective equations are solved to an accuracy of approximately $z_{ref}(\mathrm{EQ})\varepsilon_R$. (Eq. 15.652) can be written in the symbolic form:

$$\frac{1}{\varepsilon_R}\left\|\frac{\lambda x}{z^j + z_{ref}}\right\| < 1 \tag{15.653}$$

(Eq. 15.653) can also be rewritten in the equivalent form:

$$\left\|\frac{\lambda \bar{x}}{\varepsilon_R \bar{z}^j + \varepsilon_A}\right\| < 1 \tag{15.654}$$

where $\bar{z}^j = \frac{z^j}{z^*}$ and $\bar{x} = \frac{x}{z^*}$.

z^* is the normalization factor (for example, it is the intrinsic carrier density $n_i = 1.48\times10^{10}\,/\mathrm{cm}^3$ for electron and hole equations, and the thermal voltage $u_{T0} = 25.8$ mV for the Poisson equation).

The absolute error is related to the relative error through:

$$\varepsilon_A = \varepsilon_R \frac{z_{ref}}{z^*} \tag{15.655}$$

DESSIS supports two schemes for controlling the error conditions. The default scheme is based on (Eq. 15.654). The default values for the parameters ε_A are given in Section 2.10 on page 15.73. They are accessible in the `Math` section:

```
Math{ ...
   Error( Electron ) = 1e-5
   Error( Hole )     = 1e-5
}
```

The second scheme is activated with the keyword `RelErrControl` in the `Math` section and is based on (Eq. 15.652). The default values for the parameters z_{ref} are given in Section 2.10. They are accessible in the `Math` section:

```
Math{ ...
   RelErrControl
   ErrRef( Electron ) = 1e10
   ErrRef( Hole )     = 1e10
}
```

NOTE The use of the keyword `RelErrContol` is recommended, even if none of the z_{ref} parameters are actually redefined.

32.5.2 'Plugin' iterations

This is the traditional scheme, which is also known as 'Gummel iterations' in most other device simulators. Consider that there are `n` sets of nonlinear systems $g_j(z_1 \ldots z_n) = 0$. (`n` can be, for example, 3 and the sets can be the Poisson equation and two continuity equations.) This method starts with values $z_{1(1)} \ldots z_{n(1)}$ and then solves each set $g_j = 0$ separately and consecutively. One loop could be:

$$\begin{aligned} & g_1(z_1 z_2^{(i)} \ldots z_n^{(i)}) = 0 \Rightarrow z_1^{(i+1)} \\ & \ldots \\ & g_1(z_1^{(i+1)} \ldots z_{n-1}^{(i+1)} z_n) = 0 \Rightarrow z_n^{(i+1)} \end{aligned} \tag{15.656}$$

If an update (λx) of the solution between two successive plugin iterations is defined as:

$$(\lambda x) = z_j^{(i+1)} - z_j^{(i)} \tag{15.657}$$

(Eq. 15.653) or (Eq. 15.654) can be applied for convergence control in plugin iterations.

Part V Physical Model Interface

This part of the DESSIS manual contains the following chapter:

CHAPTER 33 Physical model interface

33.1 Overview

The physical model interface (PMI) provides direct access to certain models in the semiconductor transport equations. The user can provide new C++ functions to compute these models, and DESSIS loads the functions at run-time using the dynamic loader. No access to the DESSIS source code is necessary. The user can modify the following models:

- Generation–recombination rate R, compared to (Eq. 15.20)
- Avalanche generation, that is, ionization coefficient α in (Eq. 15.225)
- Electron and hole mobilities μ_n and μ_p, compared to (Eq. 15.21) and (Eq. 15.22)
- Band gap, see Chapter 5 on page 15.151
- Band-gap narrowing ΔE_g, see Section 5.2 on page 15.151
- Electron affinity, see Section 5.2
- Effective mass, see Section 5.3 on page 15.155
- Energy relaxation times τ, compared to (Eq. 15.43) to (Eq. 15.45)
- Lifetimes τ, as used in SRH recombination (see (Eq. 15.184)) and CDL recombination (see (Eq. 15.212))
- Thermal conductivity κ, compared to (Eq. 15.25)
- Heat capacity c, compared to (Eq. 15.25)
- Optical absorption, see Section 33.22 on page 15.586
- Refractive index, see Section 13.3.5 on page 15.254
- Stress, see Section 33.24 on page 15.589
- Trap space factor, see Chapter 10 on page 15.225
- Piezoelectric polarization
- Incomplete ionization, see Chapter 6 on page 15.161

A separate interface is provided to add new entries to the DESSIS current plot file, see Section 33.28 on page 15.600.

The following steps are needed to use a PMI model in a DESSIS simulation:

- A C++ subroutine must be implemented to evaluate the PMI model. Additional C++ subroutines must be written to evaluate the derivatives of the PMI model with respect to all input variables (see Section 33.2 on page 15.536).
- The `cmi` script produces a shared object file that DESSIS loads at run-time (see Section 33.3 on page 15.538).

- The `PMIPath` variable must be defined in the `File` section of the DESSIS command file. This defines the search path for the shared object files. A PMI model is activated in the `Physics` section of the DESSIS command file by specifying its name (see Section 33.4 on page 15.539).
- Parameters for PMI models can appear in the DESSIS parameter file (see Section 33.6 on page 15.542).

These steps are discussed further in the following sections. The source code for the examples is in the directory `$ISEROOT/tcad/$ISERELEASE/lib/dessis/src`.

33.2 C++ interface

For each PMI model, the user must implement a C++ subroutine to evaluate the model. Additional subroutines are necessary to evaluate the derivatives of the model with respect to all the input variables. More specifically, the user must implement a C++ class that is derived from a base class declared in the header file `PMIModels.h`. In addition, a so-called virtual constructor function must be provided, which allocates an instance of the derived class.

For example, consider the implementation of Auger recombination as a new PMI model. (The built-in Auger recombination model is discussed in Section 9.7 on page 15.212.)

In its simplest form, Auger recombination can be written as:

$$R = C \cdot (n+p) \cdot (np - n_{i,\,eff}^2) \tag{15.658}$$

where n and p are the electron and hole densities, respectively, and $n_{i,\,eff}$ is the effective intrinsic density. DESSIS needs to evaluate the value of R and the derivatives:

$$\begin{aligned}\frac{\partial R}{\partial n} &= C(np - n_{i,\,eff}^2 + (n+p)p) \\ \frac{\partial R}{\partial p} &= C(np - n_{i,\,eff}^2 + (n+p)n) \\ \frac{\partial R}{\partial n_{i,\,eff}} &= -2C(n+p)n_{i,\,eff}\end{aligned} \tag{15.659}$$

In the header file `PMIModels.h`, the following base class is defined for recombination models:

```
class PMI_Recombination : public PMI_Dessis_Interface {

public:
  PMI_Recombination (const PMI_Environment& env);
  virtual ~PMI_Recombination ();

  virtual void Compute_r
    (const double t, const double n, const double p,
     const double nie, const double f, double& r) = 0;

  virtual void Compute_drdt
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdt) = 0;

  virtual void Compute_drdn
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdn) = 0;

  virtual void Compute_drdp
```

```
      (const double t, const double n, const double p,
       const double nie, const double f, double& drdp) = 0;

    virtual void Compute_drdnie
      (const double t, const double n, const double p,
       const double nie, const double f, double& drdnie) = 0;

    virtual void Compute_drdf
      (const double t, const double n, const double p,
       const double nie, const double f, double& drdf) = 0;
  };
```

To implement a PMI model for Auger recombination, the user must declare a derived class:

```
#include "PMIModels.h"

class Auger_Recombination : public PMI_Recombination {

  double C;

public:
  Auger_Recombination (const PMI_Environment& env);
  ~Auger_Recombination ();

  void Compute_r
    (const double t, const double n, const double p,
     const double nie, const double f, double& r);

  void Compute_drdt
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdt);

  void Compute_drdn
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdn);

  void Compute_drdp
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdp);

  void Compute_drdnie
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdnie);

  void Compute_drdf
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdf);
};
```

The constructor of the derived class is invoked for each region of the device. In this example, the variable `C` is initialized from the DESSIS parameter file:

```
Auger_Recombination::
Auger_Recombination (const PMI_Environment& env) :
  PMI_Recombination (env)
{ C = InitParameter ("C", 1e-30);
}
```

If the parameter C is not found in the parameter file, a default value of 10^{-30} is used (see Section 33.6 on page 15.542). During a Newton iteration, DESSIS evaluates a PMI model for each mesh vertex. The method

`Compute_r()` computes the recombination rate for a given vertex. According to the parameter list, the recombination rate can depend on the following variables:

`t`	Lattice temperature
`n`	Electron density
`p`	Hole density
`nie`	Effective intrinsic density
`f`	Absolute value of electric field

The result of the function is stored in the parameter `r`:

```
void Auger_Recombination::
Compute_r (const double t, const double n, const double p,
           const double nie, const double f, double& r)
{ r = C * (n + p) * (n*p - nie*nie);
  if (r < 0.0) {
    r = 0.0;
  }
}
```

Besides `Compute_r()`, the user must implement other methods to compute the partial derivatives of the recombination rate with respect to the input variables `t`, `n`, `p`, `nie`, and `f`. The implementation of `Compute_drdn()` to compute the value of $\partial R / \partial n$ is:

```
void Auger_Recombination::
Compute_drdn (const double t, const double n, const double p,
const double nie, const double f, double& drdn)
{ double r = C * (n + p) * (n*p - nie*nie);
  if (r < 0.0) {
    drdn = 0.0;
  } else {
    drdn = C * ((n*p - nie*nie) + (n + p) * p);
  }
}
```

Finally, the user must provide a so-called virtual constructor function, which allocates a variable of the new class:

```
extern "C"
PMI_Recombination* new_PMI_Recombination (const PMI_Environment& env)
{ return new Auger_Recombination (env);
}
```

NOTE This function must have C linkage and exactly the same name as declared in the header file `PMIModels.h`.

33.3 Shared object code

DESSIS assumes that the shared object code corresponding to a PMI model can be found in the file `modelname.so.arch`. The base name of this file must be identical to the name of the PMI model. The extension `.arch` depends on the hardware architecture. The script `cmi`, which is also a part of the CMI, can be used to produce the shared object files (see Compact Models, Section 3.7 on page 16.109).

33.4 DESSIS command file

To load PMI models into DESSIS, the `PMIPath` search path must be defined in the `File` section of the DESSIS command file. The value of `PMIPath` consists of a sequence of directories, for example:

```
File {
   PMIPath = ". /home/joe/lib /home/mary/dessis/lib"
}
```

For each PMI model, which appears in the `Physics` section, the given directories are searched for a corresponding shared object file `modelname.so.arch`.

The PMI in DESSIS provides access to mesh-based scalar fields specified by the user. These fields must be defined on the device grid in a separate DF–ISE data file (extension `.dat`). Up to ten datasets (`PMIUserField0`, `...`, `PMIUserField9`) can be defined. DESSIS reads the user-defined fields if the corresponding file name is given in the command file:

```
File {
   PMIUserFields = "fields"
}
```

A PMI model can be activated in the `Physics` section of the DESSIS command file by specifying the name of the PMI model in the appropriate part of the `Physics` section. Examples for different types of PMI models are:

- Generation–recombination models:

  ```
  Physics {
     Recombination (pmi_model_name ...)
  }
  ```

- Avalanche generation:

  ```
  Physics {
     Recombination (Avalanche (pmi_model_name ...))
  }
  ```

- Mobility models:

  ```
  Physics {
     Mobility (
        DopingDep (pmi_model_name)
        Enormal (pmi_model_name)
        ToCurrentEnormal (pmi_model_name)
        HighFieldSaturation (pmi_model_name driving_force)
     )
  }
  ```

A PMI model name can only consist of alphanumeric characters and underscores (_). The first character must be either a letter or an underscore. A PMI model name can also be quoted as `"model_name"` to avoid conflicts with DESSIS keywords.

All the PMI models can be specified regionwise or materialwise:

```
Physics (region = "Region.1") {
...
}
Physics (material = "AlGaAs") {
...
}
```

NOTE No PMI models are available for region interfaces and material interfaces.

Certain values of PMI models can be plotted in the `Plot` section of the DESSIS command file. The following identifiers are recognized:

- Generation–recombination models:

```
Plot {
   PMIRecombination
}
```

- User-defined fields:

```
Plot {
   PMIUserField0    PMIUserField1    PMIUserField2    PMIUserField3
   PMIUserField4    PMIUserField5    PMIUserField6    PMIUserField7
   PMIUserField8    PMIUserField9
}
```

- Piezoelectric polarization:

```
Plot {
   PE_Polarization/vector  PE_Charge
}
```

The current plot PMI can be used to add entries to the DESSIS current plot file:

```
CurrentPlot {
   pmi_CurrentPlot
}
```

33.5 Run-time support

The base class `PMI_Dessis_Interface` provides run-time support for the PMI models:

```
class PMI_Dessis_Interface {
public:
  PMI_Dessis_Interface (const PMI_Environment& env);
  virtual ~PMI_Dessis_Interface ();

  const char* Name () const;

  const PMIBaseParam* ReadParameter (const char* name) const;

  double InitParameter (const char* name, double defaultvalue) const;

  int ReadDimension () const;

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

  double ReadTime () const;

  double ReadxMoleFraction () const;
  double ReadyMoleFraction () const;

  double ReadDoping (PMI_DopingSpecies species) const;

  double ReadDoping (const char* UserSpeciesName) const;
```

```
    int IsUserFieldDefined (PMI_UserFieldIndex index) const;

    double ReadUserField (PMI_UserFieldIndex index) const;

    double ReadStress (PMI_StressIndex index) const;
  };
```

The method `Name()` returns the name of the PMI model as specified in the DESSIS command file. The methods `ReadParameter()` and `InitParameter()` read the value of a parameter from the DESSIS parameter file (see Section 33.6 on page 15.542).

`ReadDimension()` returns the dimension of the problem. The functions `ReadCoordinate()` and `ReadTime()` provide the coordinates of the current vertex [µm] and the simulation time during a transient simulation [s].

The methods `ReadxMoleFraction()` and `ReadyMoleFraction()` return the x and y mole fractions, respectively.

The methods `ReadDoping(species)` and `ReadDoping(UserSpeciesName)` return the doping profiles for the current vertex [cm^{-3}]. The string `UserSpeciesName` is the same as in the file `datexcodes.txt` (see Chapter 2.14.2 on page 15.98). The enumeration type `PMI_DopingSpecies` is used to select the doping species, the incomplete ionization doping species, and their derivatives:

```
enum PMI_DopingSpecies {
// Acceptors
  PMI_Boron,
  PMI_Indium,
  PMI_PDopant,
  PMI_Acceptor,               // total acceptor concentration
// incomplete ionization entries
  PMI_AcceptorMinus,          // total incomplete ionization acceptor concentration
  PMI_AcceptorMinusPer_hDensity,
  PMI_AcceptorMinusPerT,

// Donors
  PMI_Phosphorus,
  PMI_Arsenic,
  PMI_Antimony,
  PMI_Nitrogen,
  PMI_NDopant,
  PMI_Donor,                  // total donor concentration
// incomplete ionization entries
  PMI_DonorPlus,              // total incomplete ionization donor concentration
  PMI_DonorPlusPer_eDensity,
  PMI_DonorPlusPerT
};
```

NOTE The species `PMI_Acceptor` and `PMI_Donor` are always defined. The remaining entries are only defined if they appear in the DESSIS doping input file. The incomplete ionization entries are only accessible if the option `IncompleteIonization` is activated (see Chapter 6 on page 15.161).

The method `IsUserFieldDefined()` checks if a user-defined field has been specified. The enumeration type `PMI_UserFieldIndex` selects the desired field:

```
enum PMI_UserFieldIndex {
   PMI_UserField0, PMI_UserField1, PMI_UserField2, PMI_UserField3,
   PMI_UserField4, PMI_UserField5, PMI_UserField6, PMI_UserField7,
   PMI_UserField8, PMI_UserField9
};
```

If a field is defined, the method `ReadUserField()` returns its value for the current vertex. The method `ReadStress()` returns the value of one of the following stress components:

```
enum PMI_StressIndex {
   PMI_StressXX, PMI_StressYY, PMI_StressZZ,
   PMI_StressYZ, PMI_StressXZ, PMI_StressXY
};
```

33.6 DESSIS parameter file

For each PMI model, a corresponding section with the same name can appear in the DESSIS parameter file:

```
PMI_model_name {
   par1 = value
   par2 = value
   ...
}
```

NOTE Parameter names can only consist of alphanumeric characters and underscores (_). The first character must be either a letter or an underscore.

The parameters can be specified regionwise and materialwise:

```
Region = "Region.1" {
   PMI_model_name {
   ...
   }
}
Material = "AlGaAs" {
   PMI_model_name {
   ...
   }
}
```

NOTE No user models can be specified for region interfaces and material interfaces. Therefore, PMI parameters specified for these interfaces are ignored.

The method `PMI_Dessis_Interface::ReadParameter()` can be used to obtain the value of a parameter given its name. `ReadParameter()` returns a pointer to a variable of type `PMIBaseParam`. A `NULL` pointer indicates that the parameter has not been defined in the parameter file. If the pointer `p` in:

```
const PMIBaseParam* p = ReadParameter ("name of parameter");
```

is not `NULL`, the following assignment is a valid C++ statement:

```
double d = *p;
```

The variable `d` is assigned the value of the parameter `p`. The method `PMI_Dessis_Interface::InitParameter()` checks if a parameter has been specified in the DESSIS parameter file. If the parameter has been specified, the given value is taken. Otherwise, the default value is used.

33.7 Generation–recombination model

The recombination rate R appears in the electron and hole continuity equations (see (Eq. 15.20)).

33.7.1 Dependencies

The recombination rate R may depend on these variables:

`t`	Lattice temperature [K]
`n`	Electron density [cm^{-3}]
`p`	Hole density [cm^{-3}]
`nie`	Effective intrinsic density [cm^{-3}]
`f`	Absolute value of electric field [Vcm^{-1}]

The PMI model must compute the following results:

`r`	Generation–recombination rate [$\mathrm{cm}^{-3}\mathrm{s}^{-1}$]
`drdt`	Derivative of `r` with respect to `t` [$\mathrm{cm}^{-3}\mathrm{s}^{-1}\mathrm{K}^{-1}$]
`drdn`	Derivative of `r` with respect to `n` [s^{-1}]
`drdp`	Derivative of `r` with respect to `p` [s^{-1}]
`drdnie`	Derivative of `r` with respect to `nie` [s^{-1}]
`drdf`	Derivative of `r` with respect to `f` [$\mathrm{cm}^{-2}\mathrm{s}^{-1}\mathrm{V}^{-1}$]

33.7.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Recombination : public PMI_Dessis_Interface {

public:
  PMI_Recombination (const PMI_Environment& env);
  virtual ~PMI_Recombination ();

  virtual void Compute_r
    (const double t, const double n, const double p,
     const double nie, const double f, double& r) = 0;

  virtual void Compute_drdt
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdt) = 0;

  virtual void Compute_drdn
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdn) = 0;

  virtual void Compute_drdp
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdp) = 0;

  virtual void Compute_drdnie
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdnie) = 0;

  virtual void Compute_drdf
    (const double t, const double n, const double p,
     const double nie, const double f, double& drdf) = 0;
};
```

The prototype for the virtual constructor is:

```
typedef PMI_Recombination* new_PMI_Recombination_func
   (const PMI_Environment& env);
extern "C" new_PMI_Recombination_func new_PMI_Recombination;
```

By default, DESSIS assumes that a PMI generation–recombination model depends on the electric field. However, the user can implement the optional function `PMI_Recombination_ElectricField()` to indicate whether the model depends on the electric field. If the model does not depend on the electric field (return value of 0), the method `Compute_drdf()` is not called, and the matrix assembly in DESSIS works more efficiently:

```
typedef int PMI_Recombination_ElectricField_func ();
extern "C"
PMI_Recombination_ElectricField_func PMI_Recombination_ElectricField;
```

33.7.3 Example: Auger recombination

See Section 33.2 on page 15.536.

33.8 Avalanche generation model

The generation rate due to impact ionization can be expressed as:

$$G^{||} = \alpha_n n v_n + \alpha_p p v_p \tag{15.660}$$

where α_n and α_p are the ionization coefficients for electrons and holes, respectively (compare with (Eq. 15.225)). The PMI in DESSIS allows the user to redefine the calculation of α_n and α_p.

33.8.1 Dependencies

The ionization coefficients α_n and α_p may depend on the following variables:

`F`	Driving force [Vcm^{-1}]
`t`	Lattice temperature [K]
`bg`	Band gap [eV]
`ct`	Carrier temperature [K]
`currentWoMob[3]`	Current without mobility [cm^{-4}AVs]

The parameter `ct` represents the electron temperature during the calculation of α_n and the hole temperature during the calculation of α_p.

The parameter `currentWoMob` can be used to compute anisotropic avalanche generation. Only the first d components of the vector `currentWoMob` are defined, where d is equal to the dimension of the problem. It is recommended that only the direction of the vector `currentWoMob` is taken into account, but not its magnitude.

The PMI model must compute the following results:

`alpha`	Ionization coefficient [cm^{-1}]
`dalphadF`	Derivative of `alpha` with respect to `F` [V^{-1}]
`dalphadt`	Derivative of `alpha` with respect to `t` [$\mathrm{cm}^{-1}\mathrm{K}^{-1}$]
`dalphadbg`	Derivative of `alpha` with respect to `bg` [$\mathrm{cm}^{-1}\ \mathrm{eV}^{-1}$]
`dalphadct`	Derivative of `alpha` with respect to `ct` [$\mathrm{cm}^{-1}\mathrm{K}^{-1}$]
`dalphadcurrentWoMob[3]`	Derivative of `alpha` with respect to `currentWoMob` [$\mathrm{cm}^3\mathrm{A}^{-1}\mathrm{V}^{-1}\mathrm{s}^{-1}$]

Only the first d components of the vector `dalphadcurrentWoMob` need to be computed.

33.8.2 C++ interface

Different driving forces for avalanche generation can be selected in the DESSIS command file. The enumeration type `PMI_AvalancheDrivingForce`, defined in `PMIModels.h`, is used to reflect the selection of the user:

```
enum PMI_AvalancheDrivingForce {
   PMI_AvalancheElectricField,
   PMI_AvalancheParallelElectricField,
   PMI_AvalancheGradQuasiFermi
};
```

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Avalanche : public PMI_Dessis_Interface {

private:
  const PMI_AvalancheDrivingForce drivingForce;

public:
  PMI_Avalanche (const PMI_Environment& env,
                 const PMI_AvalancheDrivingForce force);
  virtual ~PMI_Avalanche ();

  PMI_AvalancheDrivingForce AvalancheDrivingForce () const
    { return drivingForce; }

  virtual void Compute_alpha
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& alpha) = 0;

  virtual void Compute_dalphadF
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadF) = 0;

  virtual void Compute_dalphadt
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadt) = 0;

  virtual void Compute_dalphadbg
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadbg) = 0;

  virtual void Compute_dalphadct
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadct) = 0;

  virtual void Compute_dalphadcurrentWoMob
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double dalphadcurrentWoMob[3]) = 0;
};
```

Two virtual constructors are required for the calculation of the ionization coefficients α_n and α_p :

```
typedef PMI_Avalanche* new_PMI_Avalanche_func
(const PMI_Environment& env, const PMI_AvalancheDrivingForce force);
extern "C" new_PMI_Avalanche_func new_PMI_e_Avalanche;
extern "C" new_PMI_Avalanche_func new_PMI_h_Avalanche;
```

33.8.3 Example: Okuto model

Okuto and Crowell propose the following expression for the ionization coefficient α :

$$\alpha(F) = a \cdot [1 + c(T - T_0)] \cdot F e^{-\left(\frac{b[1 + d(T - T_0)]}{F}\right)^2} \qquad (15.661)$$

This built-in model is discussed in Section 9.9.3 on page 15.215 and its implementation as a PMI model is:

```
#include "PMIModels.h"

class Okuto_Avalanche : public PMI_Avalanche {

protected:
  const double T0;
  double a, b, c, d;

public:
  Okuto_Avalanche (const PMI_Environment& env,
                   const PMI_AvalancheDrivingForce force);

  ~Okuto_Avalanche ();

  void Compute_alpha
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& alpha);

  void Compute_dalphadF
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadF);

  void Compute_dalphadt
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadt);

  void Compute_dalphadbg
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadbg);
  void Compute_dalphadct
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double& dalphadct);

  void Compute_dalphadcurrentWoMob
    (const double F, const double t, const double bg,
     const double ct, const double currentWoMob[3], double dalphadcurrentWoMob[3]);
};

Okuto_Avalanche::
Okuto_Avalanche (const PMI_Environment& env,
                 const PMI_AvalancheDrivingForce force) :
  PMI_Avalanche (env, force),
  T0 (300.0)
{
}

Okuto_Avalanche::
~Okuto_Avalanche ()
{
}

void Okuto_Avalanche::
Compute_alpha (const double F, const double t, const double bg,
               const double ct, const double currentWoMob[3], double& alpha)
{ const double aa = a * (1.0 + c * (t - T0));
  const double bb = b * (1.0 + d * (t - T0)) / F;
  alpha = aa * F * exp (-bb*bb);
}

void Okuto_Avalanche::
Compute_dalphadF (const double F, const double t, const double bg,
                  const double ct, const double currentWoMob[3], double& dalphadF)
{ const double aa = a * (1.0 + c * (t - T0));
  const double bb = b * (1.0 + d * (t - T0)) / F;
  const double alpha = aa * F * exp (-bb*bb);
  dalphadF = (alpha / F) * (1.0 + 2.0*bb*bb);
}

void Okuto_Avalanche::
Compute_dalphadt (const double F, const double t, const double bg,
                  const double ct, const double currentWoMob[3], double& dalphadt)
{ const double aa = a * (1.0 + c * (t - T0));
  const double bb = b * (1.0 + d * (t - T0)) / F;
  const double tmp = F * exp (-bb*bb);
  dalphadt = tmp * (a * c - 2.0 * aa * bb * b * d / F);
}

void Okuto_Avalanche::
Compute_dalphadbg (const double F, const double t, const double bg,
                   const double ct, const double currentWoMob[3], double& dalphadbg)
{ dalphadbg = 0.0;
}

void Okuto_Avalanche::
Compute_dalphadct (const double F, const double t, const double bg,
                   const double ct, const double currentWoMob[3], double& dalphadct)
{ dalphadct = 0.0;
}

void Okuto_Avalanche::
Compute_dalphadcurrentWoMob (const double F, const double t, const double bg,
                             const double ct, const double currentWoMob[3],
                             double dalphadcurrentWoMob[3])
{ const int dim = ReadDimension ();
  for (int k = 0; k < dim; k++) {
    dalphadcurrentWoMob [k] = 0.0;
  }
}

class Okuto_e_Avalanche : public Okuto_Avalanche {

public:
  Okuto_e_Avalanche (const PMI_Environment& env,
                     const PMI_AvalancheDrivingForce force);

  ~Okuto_e_Avalanche () {}
};

Okuto_e_Avalanche::
Okuto_e_Avalanche (const PMI_Environment& env,
                   const PMI_AvalancheDrivingForce force) :
  Okuto_Avalanche (env, force)
{ // default values
  a = InitParameter ("a_e", 0.426);
  b = InitParameter ("b_e", 4.81e5);
  c = InitParameter ("c_e", 3.05e-4);
  d = InitParameter ("d_e", 6.86e-4);
}

class Okuto_h_Avalanche : public Okuto_Avalanche {

public:
  Okuto_h_Avalanche (const PMI_Environment& env,
                     const PMI_AvalancheDrivingForce force);

  ~Okuto_h_Avalanche () {}
};

Okuto_h_Avalanche::
Okuto_h_Avalanche (const PMI_Environment& env,
                   const PMI_AvalancheDrivingForce force) :
  Okuto_Avalanche (env, force)
{ // default values
  a = InitParameter ("a_h", 0.243);
  b = InitParameter ("b_h", 6.53e+5);
  c = InitParameter ("c_h", 5.35e-4);
  d = InitParameter ("d_h", 5.67e-4);
}

extern "C"
PMI_Avalanche* new_PMI_e_Avalanche
  (const PMI_Environment& env, const PMI_AvalancheDrivingForce force)
{ return new Okuto_e_Avalanche (env, force);
}

extern "C"
PMI_Avalanche* new_PMI_h_Avalanche
  (const PMI_Environment& env, const PMI_AvalancheDrivingForce force)
{ return new Okuto_h_Avalanche (env, force);
}
```

33.9 Mobility models

DESSIS supports three types of PMI mobility models:

- Doping-dependent mobility
- Mobility degradation at interfaces
- High-field saturation

PMI and built-in models can be used simultaneously. See Chapter 8 on page 15.175 for more information about how the contributions of the different models are combined. PMI mobility models support anisotropic calculations and can be evaluated along different crystallographic axes. The enumeration type:

```
enum PMI_AnisotropyType {
   PMI_Isotropic,
   PMI_Anisotropic
};
```

determines the axis. The default is isotropic mobility. If anisotropic mobilities are activated in the DESSIS command file, the PMI mobility classes are also instantiated in the anisotropic direction.

33.10 Doping-dependent mobility

A doping-dependent PMI model must account for both the constant mobility and doping-dependent mobility models discussed in Section 8.3 on page 15.176 and Section 8.4 on page 15.176.

33.10.1 Dependencies

The constant mobility and doping-dependent mobility μ_{dop} may depend on the following variables:

`t`	Lattice temperature [K]
`n`	Electron density $[\mathrm{cm}^{-3}]$
`p`	Hole density $[\mathrm{cm}^{-3}]$

The PMI model must compute the following results:

`m`	Mobility μ_{dop} $[\mathrm{cm^2V^{-1}s^{-1}}]$
`dmdn`	Derivative of μ_{dop} with respect to `n` $[\mathrm{cm^5V^{-1}s^{-1}}]$
`dmdp`	Derivative of μ_{dop} with respect to `p` $[\mathrm{cm^5V^{-1}s^{-1}}]$
`dmdt`	Derivative of μ_{dop} with respect to `t` $[\mathrm{cm^2V^{-1}s^{-1}K^{-1}}]$

In most cases, it is not necessary to compute the derivatives with respect to the dopant concentrations.

However, to model random dopant fluctuations (see Section 15.3.4 on page 15.294), the PMI model must override the functions that compute the following values:

dmdNa	Derivative of μ_{dop} with respect to the acceptor concentration $[cm^5V^{-1}s^{-1}]$
dmdNd	Derivative of μ_{dop} with respect to the donor concentration $[cm^5V^{-1}s^{-1}]$

33.10.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_DopingDepMobility : public PMI_Dessis_Interface {

private:
  const PMI_AnisotropyType anisoType;

public:
  PMI_DopingDepMobility (const PMI_Environment& env,
                         const PMI_AnisotropyType anisotype);
  virtual ~PMI_DopingDepMobility ();

  PMI_AnisotropyType AnisotropyType () const { return anisoType; }

  virtual void Compute_m
    (const double n, const double p,
     const double t, double& m) = 0;

  virtual void Compute_dmdn
    (const double n, const double p,
     const double t, double& dmdn) = 0;

  virtual void Compute_dmdp
    (const double n, const double p,
     const double t, double& dmdp) = 0;

  virtual void Compute_dmdt
    (const double n, const double p,
     const double t, double& dmdt) = 0;

virtual void Compute_dmdNa
    (const double n, const double p,
     const double t, double& dmdNa)
    { dmdNa=0.0; }

virtual void Compute_dmdNd
    (const double n, const double p,
     const double t, double& dmdNd)
    { dmdNd=0.0; }

};
```

Two virtual constructors are required for electron and hole mobilities:

```
typedef PMI_DopingDepMobility* new_PMI_DopingDepMobility_func
   (const PMI_Environment& env, const PMI_AnisotropyType anisotype);
extern "C" new_PMI_DopingDepMobility_func new_PMI_DopingDep_e_Mobility;
extern "C" new_PMI_DopingDepMobility_func new_PMI_DopingDep_h_Mobility;
```

33.10.3 Example: Masetti model

The built-in Masetti model (see Section 8.4.2 on page 15.177) can also be implemented as a PMI model:

```
#include "PMIModels.h"

class Masetti_DopingDepMobility : public PMI_DopingDepMobility {
protected:
  const double T0;
  double mumax, Exponent, mumin1, mumin2, mu1, Pc, Cr, Cs, alpha, beta;

public:
  Masetti_DopingDepMobility (const PMI_Environment& env,
                             const PMI_AnisotropyType anisotype);
  ~Masetti_DopingDepMobility () {}

  void Compute_m
    (const double n, const double p,
     const double t, double& m);

  void Compute_dmdn
    (const double n, const double p,
     const double t, double& dmdn);

  void Compute_dmdp
    (const double n, const double p,
     const double t, double& dmdp);

  void Compute_dmdt
    (const double n, const double p,
     const double t, double& dmdt);
};


Masetti_DopingDepMobility::
Masetti_DopingDepMobility (const PMI_Environment& env,
                           const PMI_AnisotropyType anisotype) :
  PMI_DopingDepMobility (env, anisotype),
  T0 (300.0)
{
}

void Masetti_DopingDepMobility::
Compute_m (const double n, const double p,
           const double t, double& m)
{ const double mu_const = mumax * pow (t/T0, -Exponent);
  const double Ni = Max (ReadDoping (PMI_Donor) +
                         ReadDoping (PMI_Acceptor), 1.0);
  m = mumin1 * exp (-Pc / Ni) +
      (mu_const - mumin2) / (1.0 + pow (Ni / Cr, alpha)) -
      mu1 / (1.0 + pow (Cs / Ni, beta));
}

void Masetti_DopingDepMobility::
Compute_dmdn (const double n, const double p,
              const double t, double& dmdn)
{ dmdn = 0.0;
}

void Masetti_DopingDepMobility::
Compute_dmdp (const double n, const double p,
              const double t, double& dmdp)
```

```
{ dmdp = 0.0;
}

void Masetti_DopingDepMobility::
Compute_dmdt (const double n, const double p,
              const double t, double& dmdt)
{ const double Ni = Max (ReadDoping (PMI_Donor) +
                         ReadDoping (PMI_Acceptor), 1.0);
  dmdt = mumax * (-Exponent/T0) * pow (t/T0, -Exponent - 1.0) /
         (1.0 + pow (Ni / Cr, alpha));
}

class Masetti_e_DopingDepMobility : public Masetti_DopingDepMobility {
public:
  Masetti_e_DopingDepMobility (const PMI_Environment& env,
                               const PMI_AnisotropyType anisotype);
  ~Masetti_e_DopingDepMobility () {}
};

Masetti_e_DopingDepMobility::
Masetti_e_DopingDepMobility (const PMI_Environment& env,
                             const PMI_AnisotropyType anisotype) :
  Masetti_DopingDepMobility (env, anisotype)

{ // default values
  mumax = InitParameter ("mumax_e", 1417.0);
  Exponent = InitParameter ("Exponent_e", 2.5);
  mumin1 = InitParameter ("mumin1_e", 52.2);
  mumin2 = InitParameter ("mumin2_e", 52.2);
  mu1 = InitParameter ("mu1_e", 43.4);
  Pc = InitParameter ("Pc_e", 0.0);
  Cr = InitParameter ("Cr_e", 9.68e16);
  Cs = InitParameter ("Cs_e", 3.43e20);
  alpha = InitParameter ("alpha_e", 0.680);
  beta = InitParameter ("beta_e", 2.0);
}

class Masetti_h_DopingDepMobility : public Masetti_DopingDepMobility {
public:
  Masetti_h_DopingDepMobility (const PMI_Environment& env,
                               const PMI_AnisotropyType anisotype);
  ~Masetti_h_DopingDepMobility () {}
};

Masetti_h_DopingDepMobility::
Masetti_h_DopingDepMobility (const PMI_Environment& env,
                             const PMI_AnisotropyType anisotype) :
  Masetti_DopingDepMobility (env, anisotype)

{ // default values
  mumax = InitParameter ("mumax_h", 470.5);
  Exponent = InitParameter ("Exponent_h", 2.2);
  mumin1 = InitParameter ("mumin1_h", 44.9);
  mumin2 = InitParameter ("mumin2_h", 0.0);
  mu1 = InitParameter ("mu1_h", 29.0);
  Pc = InitParameter ("Pc_h", 9.23e16);
  Cr = InitParameter ("Cr_h", 2.23e17);
  Cs = InitParameter ("Cs_h", 6.10e20);
  alpha = InitParameter ("alpha_h", 0.719);
  beta = InitParameter ("beta_h", 2.0);
}

extern "C"
```

```
PMI_DopingDepMobility* new_PMI_DopingDep_e_Mobility
  (const PMI_Environment& env, const PMI_AnisotropyType anisotype)
{ return new Masetti_e_DopingDepMobility (env, anisotype);
}

extern "C"
PMI_DopingDepMobility* new_PMI_DopingDep_h_Mobility
  (const PMI_Environment& env, const PMI_AnisotropyType anisotype)
{ return new Masetti_h_DopingDepMobility (env, anisotype);
}
```

33.11 Mobility degradation at interfaces

DESSIS uses Mathiessen's rule:

$$\frac{1}{\mu} = \frac{1}{\mu_{\text{dop}}} + \frac{1}{\mu_{\text{enormal}}} \tag{15.662}$$

to combine the constant and doping-dependent mobility μ_{dop}, and the surface contribution μ_{enormal} (see Section 8.5 on page 15.180). To express no mobility degradation, for example, in the bulk of a device, it is necessary to set $\mu_{\text{enormal}} = \infty$. To avoid numeric difficulties, the PMI requires the calculation of the inverse mobility $1/\mu_{\text{enormal}}$ instead of μ_{enormal}.

As an additional precaution, DESSIS does not evaluate the PMI model if the normal electric field $F_{\perp}$ is very small.

33.11.1 Dependencies

The mobility degradation at interfaces may depend on the following variables:

`dist`	Distance to nearest interface [cm]
`pot`	Electrostatic potential [V]
`enorm`	Normal electric field [Vcm^{-1}]
`t`	Lattice temperature [K]
`n`	Electron density [cm^{-3}]
`p`	Hole density [cm^{-3}]
`ct`	Carrier temperature [K]

NOTE If DESSIS cannot determine the distance to the nearest interface, the value of $\text{dist} = 10^{10}$ is used.

NOTE The carrier temperature `ct` represents the electron temperature during the evaluation of the model for electrons, and the hole temperature during the evaluation of the model for holes. The parameter `ct` is only defined for hydrodynamic simulations. Otherwise, the value of $\text{ct} = 0$ is used.

The PMI model must compute the following results:

muinv	Inverse of mobility $1/\mu_{\mathrm{enormal}}$ [cm^{-2}Vs]
dmuinvdpot	Derivative of $1/\mu_{\mathrm{enormal}}$ with respect to pot [cm^{-2}s]
dmuinvdenorm	Derivative of $1/\mu_{\mathrm{enormal}}$ with respect to enorm [cm^{-1}s]
dmuinvdn	Derivative of $1/\mu_{\mathrm{enormal}}$ with respect to n [cmVs]
dmuinvdp	Derivative of $1/\mu_{\mathrm{enormal}}$ with respect to p [cmVs]
dmuinvdt	Derivative of $1/\mu_{\mathrm{enormal}}$ with respect to t [cm^{-2}VsK^{-1}]
dmuinvdct	Derivative of $1/\mu_{\mathrm{enormal}}$ with respect to ct [cm^{-2}VsK^{-1}]

In most cases, it is not necessary to compute the derivatives with respect to the dopant concentrations. However, to model random dopant fluctuations (see Section 15.3.4 on page 15.294), the PMI model must override the functions that compute the following values:

dmuinvdNa	Derivative of $1/\mu_{\mathrm{enormal}}$ with respect to the acceptor concentration [cmVs]
dmuinvdNd	Derivative of $1/\mu_{\mathrm{enormal}}$ with respect to the donor concentration [cmVs]

33.11.2 C++ interface

The enumeration type PMI_EnormalType describes the type of the normal electric field $F_{\perp}$:

```
enum PMI_EnormalType {
   PMI_EnormalToCurrent,
   PMI_EnormalToInterface
};
```

The following base class is declared in the file PMIModels.h:

```
class PMI_EnormalMobility : public PMI_Dessis_Interface {

private:
  const PMI_EnormalType enormalType;
  const PMI_AnisotropyType anisoType;

public:
  PMI_EnormalMobility (const PMI_Environment& env,
                       const PMI_EnormalType type,
                       const PMI_AnisotropyType anisotype);

  virtual ~PMI_EnormalMobility ();

  PMI_EnormalType EnormalType () const { return enormalType; }
  PMI_AnisotropyType AnisotropyType () const { return anisoType; }

  virtual void Compute_muinv
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& muinv) = 0;

  virtual void Compute_dmuinvdpot
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdpot) = 0;

  virtual void Compute_dmuinvdenorm
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdenorm) = 0;

  virtual void Compute_dmuinvdn
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdn) = 0;

  virtual void Compute_dmuinvdp
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdp) = 0;

  virtual void Compute_dmuinvdt
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdt) = 0;

  virtual void Compute_dmuinvdct
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdct) = 0;

virtual void Compute_dmuinvdNa
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdNa)
    { dmuinvdNa=0.0; }

virtual void Compute_dmuinvdNd
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdNd)
    { dmuinvdNd=0.0; }

};
```

Two virtual constructors are required for electron and hole mobilities:

```
typedef PMI_EnormalMobility* new_PMI_EnormalMobility_func
  (const PMI_Environment& env, const PMI_EnormalType type,
   const PMI_AnisotropyType anisotype);
extern "C" new_PMI_EnormalMobility_func new_PMI_Enormal_e_Mobility;
extern "C" new_PMI_EnormalMobility_func new_PMI_Enormal_h_Mobility;
```

33.11.3 Example: Lombardi model

This example illustrates the implementation of a slightly simplified Lombardi model (see Section 8.5 on page 15.180) using the PMI. The contribution due to acoustic phonon-scattering has the form:

$$\mu_{\mathrm{ac}} = \frac{B}{F_\perp} + \frac{C N_i^\lambda}{F_\perp^{1/3}(T/T_0)} \qquad (15.663)$$

where $T_0 = 300\ \mathrm{K}$ and $N_i = N_D + N_A$ is the total concentration of ionized impurities.

The contribution due to surface roughness scattering is given by:

$$\mu_{sr} = \left(\frac{F_\perp^2}{\delta} + \frac{F_\perp^3}{\eta}\right)^{-1} \tag{15.664}$$

The mobilities μ_{ac} and μ_{sr} are combined according to Mathiessen's rule with an additional damping factor:

$$\frac{1}{\mu_{enormal}} = e^{-\frac{l}{l_{crit}}} \cdot \left(\frac{1}{\mu_{ac}} + \frac{1}{\mu_{sr}}\right) \tag{15.665}$$

where l is the distance to the nearest semiconductor–insulator interface point:

```
#include "PMIModels.h"

class Lombardi_EnormalMobility : public PMI_EnormalMobility {

protected:
  const double T0;
  double B, C, lambda, delta, eta, l_crit;

public:
  Lombardi_EnormalMobility (const PMI_Environment& env,
                            const PMI_EnormalType type,
                            const PMI_AnisotropyType anisotype);


~Lombardi_EnormalMobility ();

  void Compute_muinv
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& muinv);

  void Compute_dmuinvdpot
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdpot);

  void Compute_dmuinvdenorm
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdenorm);

  void Compute_dmuinvdn
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdn);

  void Compute_dmuinvdp
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdp);

  void Compute_dmuinvdt
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdt);

  void Compute_dmuinvdct
    (const double dist, const double pot,
     const double enorm, const double n, const double p,
     const double t, const double ct, double& dmuinvdct);
};


Lombardi_EnormalMobility::
Lombardi_EnormalMobility (const PMI_Environment& env,
                          const PMI_EnormalType type,
                          const PMI_AnisotropyType anisotype) :
  PMI_EnormalMobility (env, type, anisotype),
  T0 (300.0)
{
}

Lombardi_EnormalMobility::
~Lombardi_EnormalMobility ()
{
}

void Lombardi_EnormalMobility::
Compute_muinv (const double dist, const double pot,
               const double enorm, const double n, const double p,
               const double t, const double ct, double& muinv)
{ const double Ni = ReadDoping (PMI_Donor) + ReadDoping (PMI_Acceptor);
  const double denom_ac_inv =
    B + pow (enorm, 2.0/3.0) * C * pow (Ni, lambda) * T0 / t;
  const double mu_ac_inv = enorm / denom_ac_inv;
  const double mu_sr_inv = enorm * enorm / delta + pow (enorm, 3.0) / eta;
  const double damping = exp (-dist/l_crit);
  muinv = damping * (mu_ac_inv + mu_sr_inv);
}

void Lombardi_EnormalMobility::
Compute_dmuinvdpot (const double dist, const double pot,
                    const double enorm, const double n, const double p,
                    const double t, const double ct, double& dmuinvdpot)
{ dmuinvdpot = 0.0;
}

void Lombardi_EnormalMobility::
Compute_dmuinvdenorm (const double dist, const double pot,
                      const double enorm, const double n, const double p,
                      const double t, const double ct, double& dmuinvdenorm)
{ const double Ni = ReadDoping (PMI_Donor) + ReadDoping (PMI_Acceptor);
  const double denom_ac_inv =
    B + pow (enorm, 2.0/3.0) * C * pow (Ni, lambda) * T0 / t;
  const double dmu_ac_inv_denorm =
    (2.0 * B + denom_ac_inv) / (3.0 * denom_ac_inv * denom_ac_inv);
  const double mu_sr_inv_denorm =
    2.0 * enorm / delta + 3.0 * enorm * enorm / eta;
  const double damping = exp (-dist/l_crit);
  dmuinvdenorm = damping * (dmu_ac_inv_denorm + mu_sr_inv_denorm);
}

void Lombardi_EnormalMobility::
Compute_dmuinvdn (const double dist, const double pot,
                  const double enorm, const double n, const double p,
                  const double t, const double ct, double& dmuinvdn)
{ dmuinvdn = 0.0;
}

void Lombardi_EnormalMobility::
Compute_dmuinvdp (const double dist, const double pot,
                  const double enorm, const double n, const double p,
                  const double t, const double ct, double& dmuinvdp)
{ dmuinvdp = 0.0;
}

void Lombardi_EnormalMobility::
Compute_dmuinvdt (const double dist, const double pot,
                  const double enorm, const double n, const double p,
                  const double t, const double ct, double& dmuinvdt)
{ const double Ni = ReadDoping (PMI_Donor) + ReadDoping (PMI_Acceptor);
  const double factor = pow (enorm, 2.0/3.0) * C * pow (Ni, lambda) * T0;
  const double denom_ac_inv = B + factor / t;
  const double dmu_ac_inv_dt =
    enorm * factor / (denom_ac_inv * denom_ac_inv * t * t);
  const double damping = exp (-dist/l_crit);
  dmuinvdt = damping * dmu_ac_inv_dt;
}

void Lombardi_EnormalMobility::
Compute_dmuinvdct (const double dist, const double pot,
                   const double enorm, const double n, const double p,
                   const double t, const double ct, double& dmuinvdct)
{ dmuinvdct = 0.0;
}

class Lombardi_e_EnormalMobility : public Lombardi_EnormalMobility {
public:
  Lombardi_e_EnormalMobility (const PMI_Environment& env,
                              const PMI_EnormalType type,
                              const PMI_AnisotropyType anisotype);

  ~Lombardi_e_EnormalMobility () {}
};

Lombardi_e_EnormalMobility::
Lombardi_e_EnormalMobility (const PMI_Environment& env,
                            const PMI_EnormalType type,
                            const PMI_AnisotropyType anisotype) :
  Lombardi_EnormalMobility (env, type, anisotype)
{ // default values
  B = InitParameter ("B_e", 4.750e7);
  C = InitParameter ("C_e", 580.0);
  lambda = InitParameter ("lambda_e", 0.125);
  delta = InitParameter ("delta_e", 5.82e14);
  eta = InitParameter ("eta_e", 5.82e30);
  l_crit = InitParameter ("l_crit_e", 1.0e-6);
}

class Lombardi_h_EnormalMobility : public Lombardi_EnormalMobility {
public:
  Lombardi_h_EnormalMobility (const PMI_Environment& env,
                              const PMI_EnormalType type,
                              const PMI_AnisotropyType anisotype);

  ~Lombardi_h_EnormalMobility () {}
};

Lombardi_h_EnormalMobility::
Lombardi_h_EnormalMobility (const PMI_Environment& env,
                            const PMI_EnormalType type,
                            const PMI_AnisotropyType anisotype) :
  Lombardi_EnormalMobility (env, type, anisotype)
{ // default values
```

```
    B = InitParameter ("B_h", 9.925e6);
    C = InitParameter ("C_h", 2947.0);
    lambda = InitParameter ("lambda_h", 0.0317);
    delta = InitParameter ("delta_h", 2.0546e14);
    eta = InitParameter ("eta_h", 2.0546e30);
    l_crit = InitParameter ("l_crit_h", 1.0e-6);
  }

  extern "C"
  PMI_EnormalMobility* new_PMI_Enormal_e_Mobility
    (const PMI_Environment& env, const PMI_EnormalType type,
     const PMI_AnisotropyType anisotype)
  { return new Lombardi_e_EnormalMobility (env, type, anisotype);
  }

  extern "C"
  PMI_EnormalMobility* new_PMI_Enormal_h_Mobility
    (const PMI_Environment& env, const PMI_EnormalType type,
     const PMI_AnisotropyType anisotype)
  { return new Lombardi_h_EnormalMobility (env, type, anisotype);
  }
```

33.12 High-field saturation model

The high-field saturation model computes the final mobility μ as a function of the low-field mobility μ_{low} and the driving force F (see Section 8.8 on page 15.193).

33.12.1 Dependencies

The mobility μ computed by a high-field mobility model may depend on the following variables:

`pot`	Electrostatic potential [V]
`t`	Lattice temperature [K]
`n`	Electron density $[\mathrm{cm}^{-3}]$
`p`	Hole density $[\mathrm{cm}^{-3}]$
`ct`	Carrier temperature [K]
`mulow`	Low-field mobility μ_{low} $[\mathrm{cm}^2\mathrm{V}^{-1}\mathrm{s}^{-1}]$
`F`	Driving force $[\mathrm{Vcm}^{-1}]$

NOTE The carrier temperature `ct` represents the electron temperature during the evaluation of the model for electrons, and the hole temperature during the evaluation of the model for holes. The parameter `ct` is only defined for hydrodynamic simulations. Otherwise, the value of $\mathrm{ct} = 0$ is used.

The PMI model must compute the following results:

`mu`	Mobility μ $[\mathrm{cm}^2\mathrm{V}^{-1}\mathrm{s}^{-1}]$
`dmudpot`	Derivative of μ with respect to `pot` $[\mathrm{cm}^2\mathrm{V}^{-2}\mathrm{s}^{-1}]$

`dmudn`	Derivative of μ with respect to `n` $[\mathrm{cm^5V^{-1}s^{-1}}]$
`dmudp`	Derivative of μ with respect to `p` $[\mathrm{cm^5V^{-1}s^{-1}}]$
`dmudt`	Derivative of μ with respect to `t` $[\mathrm{cm^2V^{-1}s^{-1}K^{-1}}]$
`dmudct`	Derivative of μ with respect to `ct` $[\mathrm{cm^2V^{-1}s^{-1}K^{-1}}]$
`dmudmulow`	Derivative of μ with respect to `mulow` (1)
`dmudF`	Derivative of μ with respect to `F` $[\mathrm{cm^3V^{-2}s^{-1}}]$

In most cases, it is not necessary to compute the derivatives with respect to the dopant concentrations. However, to model random dopant fluctuations (see Section 15.3.4 on page 15.294), the PMI model must override the functions that compute the following values:

`dmudNa`	Derivative of μ with respect to the acceptor concentration $[\mathrm{cm^5V^{-1}s^{-1}}]$
`dmudNd`	Derivative of μ with respect to the donor concentration $[\mathrm{cm^5V^{-1}s^{-1}}]$

33.12.2 C++ interface

The enumeration type `PMI_HighFieldDrivingForce` describes the driving force as specified in the DESSIS command file:

```
enum PMI_HighFieldDrivingForce {
   PMI_HighFieldParallelElectricField,
   PMI_HighFieldGradQuasiFermi
};
```

The following base class is declared in the file `PMIModels.h`:

```
class PMI_HighFieldMobility : public PMI_Dessis_Interface {

private:
  const PMI_HighFieldDrivingForce drivingForce;
  const PMI_AnisotropyType anisoType;

public:
  PMI_HighFieldMobility (const PMI_Environment& env,
                         const PMI_HighFieldDrivingForce force,
                         const PMI_AnisotropyType anisotype);

  virtual ~PMI_HighFieldMobility ();

  PMI_HighFieldDrivingForce HighFieldDrivingForce () const { return drivingForce; }
  PMI_AnisotropyType AnisotropyType () const { return anisoType; }

  virtual void Compute_mu
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& mu) = 0;

  virtual void Compute_dmudpot
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudpot) = 0;

  virtual void Compute_dmudn
```

```
      (const double pot, const double n,
       const double p, const double t, const double ct,
       const double mulow, const double F, double& dmudn) = 0;

    virtual void Compute_dmudp
      (const double pot, const double n,
       const double p, const double t, const double ct,
       const double mulow, const double F, double& dmudp) = 0;

    virtual void Compute_dmudt
      (const double pot, const double n,
       const double p, const double t, const double ct,
       const double mulow, const double F, double& dmudt) = 0;

    virtual void Compute_dmudct
      (const double pot, const double n,
       const double p, const double t, const double ct,
       const double mulow, const double F, double& dmudct) = 0;

    virtual void Compute_dmudmulow
      (const double pot, const double n,
       const double p, const double t, const double ct,
       const double mulow, const double F, double& dmudmulow) = 0;

    virtual void Compute_dmudF
      (const double pot, const double n,
       const double p, const double t, const double ct,
       const double mulow, const double F, double& dmudF) = 0;

  virtual void Compute_dmudNa
      (const double pot, const double n,
       const double p, const double t, const double ct,
       const double mulow, const double F, double& dmudNa)
      { dmudNa=0.0; }

    virtual void Compute_dmudNd
      (const double pot, const double n,
       const double p, const double t, const double ct,
       const double mulow, const double F, double& dmudNd)
      { dmudNd=0.0; }
  };
```

Two virtual constructors are required for electron and hole mobilities:

```
typedef PMI_HighFieldMobility* new_PMI_HighFieldMobility_func
    (const PMI_Environment& env, const PMI_HighFieldDrivingForce force,
   const PMI_AnisotropyType anisotype);
extern "C" new_PMI_HighFieldMobility_func new_PMI_HighField_e_Mobility;
extern "C" new_PMI_HighFieldMobility_func new_PMI_HighField_h_Mobility;
```

33.12.3 Example: Canali model

This example presents the PMI implementation of the Canali model:

$$\mu = \frac{\mu_{\text{low}}}{\left[1 + \left(\frac{\mu_{\text{low}} F}{v_{\text{sat}}}\right)^{\beta}\right]^{1/\beta}} \tag{15.666}$$

where:

$$\beta = \beta_0\left(\frac{T}{T_0}\right)^{\beta_{\mathrm{exp}}} \qquad (15.667)$$

and:

$$v_{\mathrm{sat}} = v_{\mathrm{sat0}}\left(\frac{T}{T_0}\right)^{v_{sat,\,\mathrm{exp}}} \qquad (15.668)$$

The built-in Canali model is discussed in Section 8.8.2 on page 15.193.

```
#include "PMIModels.h"

class Canali_HighFieldMobility : public PMI_HighFieldMobility {

private:
  double beta, vsat, Fabs, val, valb, valb1, valb11b;
  void Compute_internal (const double t, const double mulow,
                         const double F);

protected:
  const double T0;
  double beta0, betaexp, vsat0, vsatexp;

public:
  Canali_HighFieldMobility (const PMI_Environment& env,
                            const PMI_HighFieldDrivingForce force,
                            const PMI_AnisotropyType anisotype);

  ~Canali_HighFieldMobility ();

  void Compute_mu
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F,
     double& mu);

  void Compute_dmudpot
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudpot);

  void Compute_dmudn
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudn);

  void Compute_dmudp
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudp);

  void Compute_dmudt
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudt);

  void Compute_dmudct
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudct);

  void Compute_dmudmulow
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudmulow);

  void Compute_dmudF
    (const double pot, const double n,
     const double p, const double t, const double ct,
     const double mulow, const double F, double& dmudF);
};

void Canali_HighFieldMobility::
Compute_internal (const double t, const double mulow, const double F)
{ beta = beta0 * pow (t/T0, betaexp);
  vsat = vsat0 * pow (t/T0, -vsatexp);
  Fabs = fabs (F);
  val = mulow * Fabs / vsat;
  valb = pow (val, beta);
  valb1 = 1.0 + valb;
  valb11b = pow (valb1, 1.0/beta);
}

Canali_HighFieldMobility::
Canali_HighFieldMobility (const PMI_Environment& env,
                          const PMI_HighFieldDrivingForce force,
                          const PMI_AnisotropyType anisotype) :
  PMI_HighFieldMobility (env, force, anisotype),
  T0 (300.0)
{
}

Canali_HighFieldMobility::
~Canali_HighFieldMobility ()
{
}


void Canali_HighFieldMobility::
Compute_mu (const double pot, const double n,
            const double p, const double t, const double ct,
            const double mulow, const double F, double& mu)
{ Compute_internal (t, mulow, F);
  mu = mulow / valb11b;
}

void Canali_HighFieldMobility::
Compute_dmudpot (const double pot, const double n,
                 const double p, const double t, const double ct,
                 const double mulow, const double F, double& dmudpot)
{ dmudpot = 0.0;
}

void Canali_HighFieldMobility::
Compute_dmudn (const double pot, const double n,
               const double p, const double t, const double ct,
               const double mulow, const double F, double& dmudn)
{ dmudn = 0.0;
}

void Canali_HighFieldMobility::
Compute_dmudp (const double pot, const double n,
               const double p, const double t, const double ct,
               const double mulow, const double F, double& dmudp)
{ dmudp = 0.0;
}

void Canali_HighFieldMobility::
Compute_dmudt (const double pot, const double n,
               const double p, const double t, const double ct,
               const double mulow, const double F, double& dmudt)
{ Compute_internal (t, mulow, F);
  const double mu = mulow / valb11b;
  const double dmudbeta = mu * (log (valb1) / (beta*beta) -
                                valb * log (val) / (beta * valb1));
  const double dmudvsat = (mu * valb) / (valb1 * vsat);
  const double dbetadt = beta * betaexp / t;
  const double dvsatdt = -vsat * vsatexp / t;
  dmudt = dmudbeta * dbetadt + dmudvsat * dvsatdt;
}

void Canali_HighFieldMobility::
Compute_dmudct (const double pot, const double n,
                const double p, const double t, const double ct,
                const double mulow, const double F, double& dmudct)
{ dmudct = 0.0;
}

void Canali_HighFieldMobility::
Compute_dmudmulow (const double pot, const double n,
                   const double p, const double t, const double ct,
                   const double mulow, const double F, double& dmudmulow)
{ Compute_internal (t, mulow, F);
  dmudmulow = 1.0 / (valb1 * valb11b);
}

void Canali_HighFieldMobility::
Compute_dmudF (const double pot, const double n,
               const double p, const double t, const double ct,
               const double mulow, const double F, double& dmudF)
{ Compute_internal (t, mulow, F);
  const double mu = mulow / valb11b;
  const double signF = (F >= 0.0) ? 1.0 : -1.0;
  dmudF = -mu * pow (mulow/vsat, beta) * pow (Fabs, beta-1.0) *
          signF / valb1;
}

class Canali_e_HighFieldMobility : public Canali_HighFieldMobility {
public:
  Canali_e_HighFieldMobility (const PMI_Environment& env,
                              const PMI_HighFieldDrivingForce force,
                              const PMI_AnisotropyType anisotype);

  ~Canali_e_HighFieldMobility () {}
};

Canali_e_HighFieldMobility::
Canali_e_HighFieldMobility (const PMI_Environment& env,
                            const PMI_HighFieldDrivingForce force,
                            const PMI_AnisotropyType anisotype) :
  Canali_HighFieldMobility (env, force, anisotype)
{ // default values
  beta0 = InitParameter ("beta0_e", 1.109);
  betaexp = InitParameter ("betaexp_e", 0.66);
  vsat0 = InitParameter ("vsat0_e", 1.07e7);
  vsatexp = InitParameter ("vsatexp_e", 0.87);
}

class Canali_h_HighFieldMobility : public Canali_HighFieldMobility {
public:
  Canali_h_HighFieldMobility (const PMI_Environment& env,
                              const PMI_HighFieldDrivingForce force,
                              const PMI_AnisotropyType anisotype);

  ~Canali_h_HighFieldMobility () {}
};

Canali_h_HighFieldMobility::
Canali_h_HighFieldMobility (const PMI_Environment& env,
                            const PMI_HighFieldDrivingForce force,
                            const PMI_AnisotropyType anisotype) :
  Canali_HighFieldMobility (env, force, anisotype)
{ // default values
  beta0 = InitParameter ("beta0_h", 1.213);
  betaexp = InitParameter ("betaexp_h", 0.17);
  vsat0 = InitParameter ("vsat0_h", 8.37e6);
  vsatexp = InitParameter ("vsatexp_h", 0.52);
}

extern "C"
PMI_HighFieldMobility* new_PMI_HighField_e_Mobility
  (const PMI_Environment& env, const PMI_HighFieldDrivingForce force,
   const PMI_AnisotropyType anisotype)
{ return new Canali_e_HighFieldMobility (env, force, anisotype);
}

extern "C"
PMI_HighFieldMobility* new_PMI_HighField_h_Mobility
  (const PMI_Environment& env, const PMI_HighFieldDrivingForce force,
   const PMI_AnisotropyType anisotype)
{ return new Canali_h_HighFieldMobility (env, force, anisotype);
}
```

33.13 Band gap

DESSIS provides a PMI to compute the energy band gap E_g in a semiconductor. It can be specified in the `Physics` section of the command file, for example:

```
Physics {
   EffectiveIntrinsicDensity (
      BandGap (pmi_model_name)
   )
}
```

The default DESSIS band gap model is selected explicitly by the keyword `Default`:

```
Physics {
   EffectiveIntrinsicDensity (
      BandGap (Default)
   )
}
```

33.13.1 Dependencies

The band gap E_g may depend on:

`t`	Lattice temperature [K]

The PMI model must compute the following results:

`bg`	Band gap E_g [eV]
`dbgdt`	Derivative of `bg` with respect to `t` [eVK^{-1}]

33.13.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_BandGap : public PMI_Dessis_Interface {

public:
  PMI_BandGap (const PMI_Environment& env);
  virtual ~PMI_BandGap ();

  virtual void Compute_bg
    (const double t, double& bg) = 0;

  virtual void Compute_dbgdt
    (const double t, double& dbgdt) = 0;
};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_BandGap* new_PMI_BandGap_func
    (const PMI_Environment& env);
extern "C" new_PMI_BandGap_func new_PMI_BandGap;
```

33.13.3 Example: Default band gap model

DESSIS uses the following default band gap model:

$$E_g(t) = E_g(0) - \frac{\alpha t^2}{t + \beta} \qquad (15.669)$$

$E_g(0)$ denotes the band gap at 0 K.

```
#include "PMIModels.h"

class Default_BandGap : public PMI_BandGap {

private:
  double Eg0, alpha, beta;

public:
  Default_BandGap (const PMI_Environment& env);
  ~Default_BandGap ();

  void Compute_bg (const double t, double& bg);

  void Compute_dbgdt (const double t, double& dbgdt);
};

Default_BandGap::
Default_BandGap (const PMI_Environment& env) :
  PMI_BandGap (env)
{ Eg0 = InitParameter ("Eg0", 1.16964);
  alpha = InitParameter ("alpha", 4.73e-4);
  beta = InitParameter ("beta", 636);
}

Default_BandGap::
~Default_BandGap ()
{
}

void Default_BandGap::
Compute_bg (const double t, double& bg)
{ bg = Eg0 - alpha * t * t / (t + beta);
}

void Default_BandGap::
Compute_dbgdt (const double t, double& dbgdt)
{ dbgdt = - alpha * t * (t + 2.0 * beta) / ((t + beta) * (t + beta));
}
extern "C"
PMI_BandGap* new_PMI_BandGap
  (const PMI_Environment& env)
{ return new Default_BandGap (env);
}
```

33.14 Band-gap narrowing

DESSIS provides a PMI to compute band-gap narrowing (see Section 5.2 on page 15.151). A user model is activated with the keyword `EffectiveIntrinsicDensity` in the `Physics` section of the command file:

```
Physics {
   EffectiveIntrinsicDensity (pmi_model_name)
}
```

33.14.1 Dependencies

A PMI band-gap narrowing model has no explicit dependencies. However, it can depend on doping concentrations through the run-time support.

The PMI model must compute:

`bgn`	Band-gap narrowing ΔE_g [eV]

In most cases, it is not necessary to compute the derivatives with respect to the dopant concentrations. However, to model random dopant fluctuations (see Section 15.3.4 on page 15.294), the PMI model must override the functions that compute the following values:

dbgndNa	Derivative of ΔE_g with respect to the acceptor concentration $[\mathrm{cm}^3\mathrm{eV}]$
dbgndNd	Derivative of ΔE_g with respect to the donor concentration $[\mathrm{cm}^3\mathrm{eV}]$

33.14.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_BandGapNarrowing : public PMI_Dessis_Interface {
public:
  PMI_BandGapNarrowing (const PMI_Environment& env);
       virtual ~PMI_BandGapNarrowing ();
       virtual void Compute_bgn
    (double& bgn) = 0;
       virtual void Compute_dbgndNa
    (double& dbgndNa){ dbgndNa=0.0; }
       virtual void Compute_dbgndNd
    (double& dbgndNd){ dbgndNd=0.0; }
};
```

The following virtual constructor must be implemented:

```
typedef PMI_BandGapNarrowing* new_PMI_BandGapNarrowing_func
    (const PMI_Environment& env);
extern "C" new_PMI_BandGapNarrowing_func new_PMI_BandGapNarrowing;
```

33.14.3 Example: Default model

The default band-gap narrowing model in DESSIS (Bennett Wilson) is given by:

$$\Delta E_g = \begin{cases} E_{\mathrm{bgn}}\left[\log\dfrac{N_i}{N_{\mathrm{ref}}}\right]^2, & N_i > N_{\mathrm{ref}}, \\ 0, & N_i \le N_{\mathrm{ref}}. \end{cases} \qquad (15.670)$$

$N_i = N_A + N_D$ denotes the total doping concentration (see Section 5.2 on page 15.151).

This model can be implemented as a PMI model as follows:

```
#include "PMIModels.h"

class Bennett_BandGapNarrowing : public PMI_BandGapNarrowing {

private:
  double Ebgn, Nref;

public:
  Bennett_BandGapNarrowing (const PMI_Environment& env);

  ~Bennett_BandGapNarrowing ();
  void Compute_bgn (double& bgn);

};

Bennett_BandGapNarrowing::
Bennett_BandGapNarrowing (const PMI_Environment& env) :
  PMI_BandGapNarrowing (env)
{ Ebgn = InitParameter ("Ebgn", 6.84e-3);
  Nref = InitParameter ("Nref", 3.162e18);
}

Bennett_BandGapNarrowing::
~Bennett_BandGapNarrowing ()
{
}

void Bennett_BandGapNarrowing::
Compute_bgn (double& bgn)
{ const double Na = ReadDoping (PMI_Acceptor);
  const double Nd = ReadDoping (PMI_Donor);
  const double Ni = Na + Nd;
  if (Ni > Nref) {
    const double tmp = log (Ni / Nref);
    bgn = Ebgn * tmp * tmp;
  } else {
    bgn = 0.0;
  }
}

extern "C"
PMI_BandGapNarrowing* new_PMI_BandGapNarrowing
  (const PMI_Environment& env)
{ return new Bennett_BandGapNarrowing (env);
}
```

33.15 Apparent band-edge shift

The apparent band-edge shift Λ_{PMI} is a quantity similar to band-gap narrowing. In contrast to band-gap narrowing, the apparent band-edge shift can depend on the solution variables (electron and hole densities, lattice temperature, and electric field). Conversely, the apparent band-edge shift does not take effect in all situations where a real band-edge shift takes effect (this is why the band-edge shift is called ‘apparent’).

Implementationwise, the apparent band-edge shift is an extension of the density gradient model (see Section 7.4 on page 15.172). For the PMI model, this implies:

- DESSIS applies the apparent band-edge shift Λ_{PMI} everywhere where it applies quantization corrections.
- By default, the apparent band-edge shift that DESSIS computes is not equal to Λ_{PMI}, but contains contributions from quantization. To remove them, set $\gamma = 0$ (see Section 7.4).
- Apart from a specification in the `Physics` section, it is necessary to specify additional equations in the `Solve` section (see Section 7.4.2 on page 15.173).
- At contacts, boundary conditions override the actual model. For example, at Ohmic contacts, the apparent band-edge shift is always zero.

To avoid problems with the boundary conditions, formulate the apparent band-edge shift as a correction to the usual (only doping-dependent) band-gap narrowing model and ensure that this correction vanishes at Ohmic contacts (see Section 4.5.1.1 on page 15.138).

The same models for Λ_{PMI} can be used for the shift of the conduction and valence bands. A positive value of Λ_{PMI} means that the band shifts outwards, away from midgap (therefore, the band gap widens).

33.15.1 Dependencies

The apparent band-edge shift Λ_{PMI} may depend on:

`n`	Electron density $[\mathrm{cm}^{-3}]$
`p`	Hole density $[\mathrm{cm}^{-3}]$
`t`	Lattice temperature $[\mathrm{K}]$
`F`	Absolute value of the electric field $[\mathrm{Vcm}^{-1}]$

The PMI model must compute the following values:

`shift`	Apparent band-edge shift Λ_{PMI} $[\mathrm{eV}]$
`dshiftdn`	Derivative of Λ_{PMI} with respect to `n` $[\mathrm{eVcm}^3]$
`dshiftdp`	Derivative of Λ_{PMI} with respect to `p` $[\mathrm{eVcm}^3]$
`dshiftdt`	Derivative of Λ_{PMI} with respect to `t` $[\mathrm{eVK}^{-1}]$
`dshiftdf`	Derivative of Λ_{PMI} with respect to `f` $[\mathrm{eVcmV}^{-1}]$

33.15.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_ApparentBandEdgeShift : public PMI_Dessis_Interface {

public:
  PMI_ApparentBandEdgeShift (const PMI_Environment& env);
  virtual ~PMI_ApparentBandEdgeShift ();

  virtual void Compute_shift
    (const double n, const double p,
     const double t, const double f,
     double& shift) = 0;

  virtual void Compute_dshiftdn
    (const double n, const double p,
     const double t, const double f,
     double& dshiftdn) = 0;

  virtual void Compute_dshiftdp
    (const double n, const double p,
     const double t, const double f,
     double& dshiftdp) = 0;
  virtual void Compute_dshiftdt
    (const double n, const double p,
     const double t, const double f,
     double& dshiftdt) = 0;

  virtual void Compute_dshiftdf
    (const double n, const double p,
     const double t, const double f,
     double& dshiftdf) = 0;
};
```

The following virtual constructor must be implemented:

```
typedef PMI_ApparentBandEdgeShift* new_PMI_ApparentBandEdgeShift_func
   (const PMI_Environment& env);
extern "C" new_PMI_ApparentBandEdgeShift_func new_PMI_ApparentBandEdgeShift;
```

33.16 Electron affinity

The electron affinity χ, that is, the energy separation between the conduction band and vacuum level, can be specified by using a PMI. The syntax in the DESSIS command file is:

```
Physics {
   Affinity (pmi_model_name)
}
```

The default DESSIS affinity model can be selected explicitly by the keyword `Default`:

```
Physics {
   Affinity (Default)
}
```

33.16.1 Dependencies

The electron affinity χ may depend on:

`t`	Lattice temperature [K]

The PMI model must compute:

`affinity`	Electron affinity χ [eV]
`affinitydt`	Derivative of `affinity` with respect to `t` [eVK^{-1}]

33.16.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Affinity : public PMI_Dessis_Interface {

public:
  PMI_Affinity (const PMI_Environment& env);
  virtual ~PMI_Affinity ();
  virtual void Compute_affinity
    (const double t, double& affinity) = 0;

  virtual void Compute_daffinitydt
    (const double t, double& daffinitydt) = 0;
};
```

The prototype for the virtual constructor is:

```
typedef PMI_Affinity* new_PMI_Affinity_func
   (const PMI_Environment& env);
extern "C" new_PMI_Affinity_func new_PMI_Affinity;
```

33.16.3 Example: Default affinity model

By default, DESSIS uses this formula to compute χ:

$$\chi(t) = \chi(0) + 0.5\frac{\alpha t^2}{t + \beta} \qquad (15.671)$$

$\chi(0)$ denotes the affinity at 0 K.

```
#include "PMIModels.h"

class Default_Affinity : public PMI_Affinity {

private:
  double Affinity0, alpha, beta;

public:
  Default_Affinity (const PMI_Environment& env);

  ~Default_Affinity ();

  void Compute_affinity (const double t, double& affinity);

  void Compute_daffinitydt (const double t, double& daffinitydt);

};

Default_Affinity::
Default_Affinity (const PMI_Environment& env) :
  PMI_Affinity (env)
{ Affinity0 = InitParameter ("Affinity0", 4.05);
  alpha = InitParameter ("alpha", 4.73e-4);
  beta = InitParameter ("beta", 636);
}

Default_Affinity::
~Default_Affinity ()
{
}

void Default_Affinity::
Compute_affinity (const double t, double& affinity)
{ affinity = Affinity0 + 0.5 * alpha * t * t / (t + beta);
}

void Default_Affinity::
```

```
Compute_daffinitydt (const double t, double& daffinitydt)
{ daffinitydt = 0.5 * alpha * t * (t + 2.0 * beta) /
                ((t + beta) * (t + beta));
}
extern "C"
PMI_Affinity* new_PMI_Affinity
  (const PMI_Environment& env)
{ return new Default_Affinity (env);
}
```

33.17 Effective mass

DESSIS provides a PMI to compute the effective mass of electrons and holes. The effective mass is always expressed as a multiple of the electron mass in vacuum. The name of the PMI model must appear in the `Physics` section of the command file:

```
Physics {
   EffectiveMass (pmi_model_name)
}
```

33.17.1 Dependencies

The relative effective mass may depend on the following variables:

`t`	Lattice temperature [K]
`bg`	Band gap [eV]

The PMI model must compute the following results:

`m`	Relative effective mass (1)
`dmdt`	Derivative of `m` with respect to `t` [K^{-1}]
`dmdbg`	Derivative of `m` with respect to `bg` [eV^{-1}]

33.17.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_EffectiveMass : public PMI_Dessis_Interface {

public:
  PMI_EffectiveMass (const PMI_Environment& env);
  virtual ~PMI_EffectiveMass ();

  virtual void Compute_m
    (const double t, const double bg, double& m) = 0;

  virtual void Compute_dmdt
    (const double t, const double bg, double& dmdt) = 0;
virtual void Compute_dmdbg
(const double t, const double bg, double& dmdbg) = 0;
};
```

Two virtual constructors are necessary to compute the effective mass of electrons and holes:

```
typedef PMI_EffectiveMass* new_PMI_EffectiveMass_func
   (const PMI_Environment& env);
extern "C" new_PMI_EffectiveMass_func new_PMI_e_EffectiveMass;
extern "C" new_PMI_EffectiveMass_func new_PMI_h_EffectiveMass;
```

33.17.3 Example: Linear effective mass model

A simple, linear effective mass model is given by:

$$m = m_{300} + \frac{dm}{dt}(t-300) \qquad (15.672)$$

m_{300} denotes the mass at 300 K. It can be implemented as follows:

```
#include "PMIModels.h"

class Linear_EffectiveMass : public PMI_EffectiveMass {

protected:
  double mass_300, dmass_dt;

public:
  Linear_EffectiveMass (const PMI_Environment& env);

  ~Linear_EffectiveMass ();

  void Compute_m (const double t, const double bg, double& m);

  void Compute_dmdt (const double t, const double bg, double& dmdt);

  void Compute_dmdbg (const double t, const double bg, double& dmdbg);
};

Linear_EffectiveMass::
Linear_EffectiveMass (const PMI_Environment& env) :
  PMI_EffectiveMass (env)
{
}

Linear_EffectiveMass::
~Linear_EffectiveMass ()
{
}

void Linear_EffectiveMass::
Compute_m (const double t, const double bg, double& m)
{ m = mass_300 + dmass_dt * (t - 300.0);
}

void Linear_EffectiveMass::
Compute_dmdt (const double t, const double bg, double& dmdt)
{ dmdt = dmass_dt;
}
```

```
void Linear_EffectiveMass::
Compute_dmdbg (const double t, const double bg, double& dmdbg)
{ dmdbg = 0.0;
}

class Linear_e_EffectiveMass : public Linear_EffectiveMass {

public:
  Linear_e_EffectiveMass (const PMI_Environment& env);

  ~Linear_e_EffectiveMass () {}

};

Linear_e_EffectiveMass::
Linear_e_EffectiveMass (const PMI_Environment& env) :
  Linear_EffectiveMass (env)
{ mass_300 = InitParameter ("mass_e_300", 1.09);
  dmass_dt = InitParameter ("dmass_e_dt", 1.6e-4);
}

class Linear_h_EffectiveMass : public Linear_EffectiveMass {

public:
  Linear_h_EffectiveMass (const PMI_Environment& env);

  ~Linear_h_EffectiveMass () {}

};

Linear_h_EffectiveMass::
Linear_h_EffectiveMass (const PMI_Environment& env) :
  Linear_EffectiveMass (env)
{ mass_300 = InitParameter ("mass_h_300", 1.15);
  dmass_dt = InitParameter ("dmass_h_dt", 9.2e-4);
}

extern "C"
PMI_EffectiveMass* new_PMI_e_EffectiveMass
  (const PMI_Environment& env)
{ return new Linear_e_EffectiveMass (env);
}

extern "C"
PMI_EffectiveMass* new_PMI_h_EffectiveMass
  (const PMI_Environment& env)
{ return new Linear_h_EffectiveMass (env);
}
```

33.18 Energy relaxation times

The model for the energy relaxation times τ in (Eq. 15.43) and (Eq. 15.44) can be specified in the `Physics` section of the DESSIS command file. The four available possibilities are:

```
Physics {
   EnergyRelaxationTimes (
      formula
      constant
```

```
        irrational
        pmi_model_name
    )
}
```

These entries have the following meaning:

`formula`	Use the value of `formula` in the DESSIS parameter file (default)
`constant`	Use constant energy relaxation times (`formula = 1`)
`irrational`	Use the ratio of two irrational polynomials (`formula = 2`)
`pmi_model_name`	Call a PMI model to compute the energy relaxation times

33.18.1 Dependencies

The energy relaxation time τ may depend on the variable:

`ct`	Carrier temperature [K]

NOTE The parameter `ct` represents the electron temperature during the calculation of τ_n and the hole temperature during the calculation of τ_p.

The PMI model must compute the following results:

`tau`	Energy relaxation time τ [s]
`dtaudct`	Derivative of τ with respect to `ct` [sK^{-1}]

33.18.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_EnergyRelaxationTime : public PMI_Dessis_Interface {

public:
  PMI_EnergyRelaxationTime (const PMI_Environment& env);

  virtual ~PMI_EnergyRelaxationTime ();

  virtual void Compute_tau
    (const double ct, double& tau) = 0;

  virtual void Compute_dtaudct
    (const double ct, double& dtaudct) = 0;

};
```

The following two virtual constructors must be implemented for electron and hole energy relaxation times:

```
typedef PMI_EnergyRelaxationTime* new_PMI_EnergyRelaxationTime_func
   (const PMI_Environment& env);
```

```
extern "C" new_PMI_EnergyRelaxationTime_func new_PMI_e_EnergyRelaxationTime;
extern "C" new_PMI_EnergyRelaxationTime_func new_PMI_h_EnergyRelaxationTime;
```

33.18.3 Example: Constant energy relaxation times

The following C++ code implements constant energy relaxation times:

```
#include "PMIModels.h"

class Const_EnergyRelaxationTime : public PMI_EnergyRelaxationTime {

protected:
  double tau_const;

public:
  Const_EnergyRelaxationTime (const PMI_Environment& env);

  ~Const_EnergyRelaxationTime ();

  void Compute_tau
    (const double ct, double& tau);

  void Compute_dtaudct
    (const double ct, double& dtaudct);

};

Const_EnergyRelaxationTime::
Const_EnergyRelaxationTime (const PMI_Environment& env) :
  PMI_EnergyRelaxationTime (env)
{
}

Const_EnergyRelaxationTime::
~Const_EnergyRelaxationTime ()
{
}

void Const_EnergyRelaxationTime::
Compute_tau (const double ct, double& tau)
{ tau = tau_const;
}

void Const_EnergyRelaxationTime::
Compute_dtaudct (const double ct, double& dtaudct)
{ dtaudct = 0.0;
}

class Const_e_EnergyRelaxationTime : public Const_EnergyRelaxationTime {

public:
  Const_e_EnergyRelaxationTime (const PMI_Environment& env);

  ~Const_e_EnergyRelaxationTime () {}

};

Const_e_EnergyRelaxationTime::
Const_e_EnergyRelaxationTime (const PMI_Environment& env) :
  Const_EnergyRelaxationTime (env)
{ tau_const = InitParameter ("tau_const_e", 0.3e-12);
}

class Const_h_EnergyRelaxationTime : public Const_EnergyRelaxationTime {

public:
  Const_h_EnergyRelaxationTime (const PMI_Environment& env);

  ~Const_h_EnergyRelaxationTime () {}

};

Const_h_EnergyRelaxationTime::
Const_h_EnergyRelaxationTime (const PMI_Environment& env) :
  Const_EnergyRelaxationTime (env)

{ tau_const = InitParameter ("tau_const_h", 0.25e-12);
}

extern "C"
PMI_EnergyRelaxationTime* new_PMI_e_EnergyRelaxationTime
  (const PMI_Environment& env)
{ return new Const_e_EnergyRelaxationTime (env);
}

extern "C"
PMI_EnergyRelaxationTime* new_PMI_h_EnergyRelaxationTime
  (const PMI_Environment& env)
{ return new Const_h_EnergyRelaxationTime (env);
}
```

33.19 Lifetimes

This PMI provides access to the electron and hole lifetimes, τ_n and τ_p, in the SRH recombination (see (Eq. 15.184)) and the coupled defect level (CDL) recombination (see (Eq. 15.212)). In the DESSIS command file, the names of the lifetime models are given as arguments to the `SRH` or `CDL` keywords:

```
Physics {
   Recombination (SRH (pmi_model_name))
}
```

or:

```
Physics {
   Recombination (CDL (pmi_model_name))
}
```

NOTE A PMI model overrides all other keywords in an `SRH` or a `CDL` statement.

33.19.1 Dependencies

A PMI lifetime model may depend on the variable:

`t`	Lattice temperature [K]

It must compute the following results:

`tau`	Lifetime τ [s]
`dtaudt`	Derivative of τ with respect to lattice temperature [sK^{-1}]

33.19.2 C++ interface

The enumeration type `PMI_LifetimeModel` describes where the PMI lifetime is used:

```
enum PMI_LifetimeModel {
   PMI_SRH,
   PMI_CDL1,
   PMI_CDL2
};
```

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Lifetime : public PMI_Dessis_Interface {

private:
  const PMI_LifetimeModel lifetimeModel;

public:
  PMI_Lifetime (const PMI_Environment& env,
                const PMI_LifetimeModel model);

  virtual ~PMI_Lifetime ();

  PMI_LifetimeModel LifetimeModel () const { return lifetimeModel; }

  virtual void Compute_tau
    (const double t, double& tau) = 0;

  virtual void Compute_dtaudt
    (const double t, double& dtaudt) = 0;
};
```

Two virtual constructors must be implemented for electron and hole lifetimes:

```
typedef PMI_Lifetime* new_PMI_Lifetime_func
   (const PMI_Environment& env, const PMI_LifetimeModel model);
extern "C" new_PMI_Lifetime_func new_PMI_e_Lifetime;
extern "C" new_PMI_Lifetime_func new_PMI_h_Lifetime;
```

33.19.3 Example: Doping- and temperature-dependent lifetimes

The following example combines doping-dependent lifetimes (Scharfetter) and temperature dependence (power law):

$$\tau = \left(\tau_{\min} + \frac{\tau_{\max} - \tau_{\min}}{1 + \left(\frac{N_A + N_D}{N_{\text{ref}}}\right)^{\gamma}}\right)\left(\frac{T}{T_0}\right)^{\alpha} \tag{15.673}$$

N_A and N_D denote acceptor and donor concentrations, respectively, and $T_0 = 300$ K:

```
#include "PMIModels.h"

class Scharfetter_Lifetime : public PMI_Lifetime {

protected:
  const double T0;
  double taumin, taumax, Nref, gamma, Talpha;

public:
  Scharfetter_Lifetime (const PMI_Environment& env,
                        const PMI_LifetimeModel model);

  ~Scharfetter_Lifetime ();
  void Compute_tau
    (const double t, double& tau);

  void Compute_dtaudt
    (const double t, double& dtaudt);

};

Scharfetter_Lifetime::
Scharfetter_Lifetime (const PMI_Environment& env,
                      const PMI_LifetimeModel model) :
  PMI_Lifetime (env, model),
  T0 (300.0)
{
}

Scharfetter_Lifetime::
~Scharfetter_Lifetime ()
{
}

void Scharfetter_Lifetime::
Compute_tau (const double t, double& tau)
{ const double Ni = ReadDoping (PMI_Acceptor) + ReadDoping (PMI_Donor);
  tau = taumin + (taumax - taumin) / (1.0 + pow (Ni/Nref, gamma));
  tau *= pow (t/T0, Talpha);
}

void Scharfetter_Lifetime::
Compute_dtaudt (const double t, double& dtaudt)
{ const double Ni = ReadDoping (PMI_Acceptor) + ReadDoping (PMI_Donor);
  dtaudt = taumin + (taumax - taumin) / (1.0 + pow (Ni/Nref, gamma));
  dtaudt *= (Talpha/T0) * pow (t/T0, Talpha-1.0);
}

class Scharfetter_e_Lifetime : public Scharfetter_Lifetime {

public:
  Scharfetter_e_Lifetime (const PMI_Environment& env,
                          const PMI_LifetimeModel model);

  ~Scharfetter_e_Lifetime () {}

};
Scharfetter_e_Lifetime::
Scharfetter_e_Lifetime (const PMI_Environment& env,
                        const PMI_LifetimeModel model) :
  Scharfetter_Lifetime (env, model)
```

```
{ taumin = InitParameter ("taumin_e", 0.0);
  taumax = InitParameter ("taumax_e", 1.0e-5);
  Nref = InitParameter ("Nref_e", 1.0e16);
  gamma = InitParameter ("gamma_e", 1.0);
  Talpha = InitParameter ("Talpha_e", -1.5);
}

class Scharfetter_h_Lifetime : public Scharfetter_Lifetime {

public:
  Scharfetter_h_Lifetime (const PMI_Environment& env,
                          const PMI_LifetimeModel model);

  ~Scharfetter_h_Lifetime () {}

};

Scharfetter_h_Lifetime::
Scharfetter_h_Lifetime (const PMI_Environment& env,
                        const PMI_LifetimeModel model) :
  Scharfetter_Lifetime (env, model)

{ taumin = InitParameter ("taumin_h", 0.0);
  taumax = InitParameter ("taumax_h", 3.0e-6);
  Nref = InitParameter ("Nref_h", 1.0e16);
  gamma = InitParameter ("gamma_h", 1.0);
  Talpha = InitParameter ("Talpha_h", -1.5);
}

extern "C"
PMI_Lifetime* new_PMI_e_Lifetime
  (const PMI_Environment& env, const PMI_LifetimeModel model)
{ return new Scharfetter_e_Lifetime (env, model);
}

extern "C"
PMI_Lifetime* new_PMI_h_Lifetime
  (const PMI_Environment& env, const PMI_LifetimeModel model)
{ return new Scharfetter_h_Lifetime (env, model);
}
```

33.20 Thermal conductivity

The PMI provides access to the lattice thermal conductivity κ in (Eq. 15.25). The model used for the evaluation of κ can be specified within the `Physics` section of the DESSIS command file. The following possibilities are available:

```
Physics {
   ThermalConductivity (
      formula
      TemperatureDependent Conductivity
      Constant Conductivity
      TemperatureDependent Resistivity
      Constant Resistivity
      pmi_model_name
   )
}
```

These entries have the following meaning:

`formula`	Use the built-in strategy (default)
`TemperatureDependent Conductivity`	Use (Eq. 15.528) (`formula = 1`)
`Constant Conductivity`	Use constant conductivity (`formula = 1`)
`TemperatureDependent Resistivity`	Use temperature-dependent resistivity (`formula = 0`)
`Constant Resistivity`	Use constant resistivity (`formula = 0`)
`pmi_model_name`	Call a PMI model to compute the thermal conductivity

The PMI supports anisotropic thermal conductivity, and the model can be evaluated along different crystallographic axes. The enumeration type `PMI_AnisotropyType` as defined in Section 33.9 on page 15.549 determines the axis. The default is isotropic thermal conductivity. If anisotropic thermal conductivity is activated in the DESSIS command file, the PMI class `PMI_ThermalConductivity` is also instantiated in the anisotropic direction.

33.20.1 Dependencies

The thermal conductivity κ may depend on the variable:

`t`	Lattice temperature [K]

The PMI model must compute the following results:

`kappa`	Thermal conductivity κ [$\mathrm{Wcm^{-1}K^{-1}}$]
`dkappadt`	derivative of κ with respect to `t` [$\mathrm{Wcm^{-1}K^{-2}}$]

33.20.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_ThermalConductivity : public PMI_Dessis_Interface {

public:
  PMI_ThermalConductivity (const PMI_Environment& env, const PMI_AnisotropyType anisotype);

  virtual ~PMI_ThermalConductivity ();

  PMI_AnisotropyType AnisotropyType () const { return anisoType; }

  virtual void Compute_kappa
    (const double t, double& kappa) = 0;

  virtual void Compute_dkappadt
    (const double t, double& dkappadt) = 0;

};
```

The following virtual constructor must be implemented:

```
typedef PMI_ThermalConductivity* new_PMI_ThermalConductivity_func
   (const PMI_Environment& env, const PMI_AnisotropyType anisotype);
extern "C" new_PMI_ThermalConductivity_func
new_PMI_ThermalConductivity;
```

33.20.3 Example: Temperature-dependent thermal conductivity

The following C++ code implements the temperature-dependent thermal conductivity:

$$\kappa(T) = \frac{1}{a + bT + cT^2} \tag{15.674}$$

as given in (Eq. 15.528):

```
#include "PMIModels.h"

class TempDep_ThermalConductivity : public PMI_ThermalConductivity {
private:
  double a, b, c;

public:
  TempDep_ThermalConductivity (const PMI_Environment& env, const PMI_AnisotropyType anisotype);
  ~TempDep_ThermalConductivity ();

  void Compute_kappa
    (const double t, double& kappa);

  void Compute_dkappadt
    (const double t, double& dkappadt);
};


TempDep_ThermalConductivity::
TempDep_ThermalConductivity (const PMI_Environment& env, const PMI_AnisotropyType anisotype) :
  PMI_ThermalConductivity (env, anisotype)
{ // default values
  a = InitParameter ("a", 0.03);
  b = InitParameter ("b", 1.56e-03);
  c = InitParameter ("c", 1.65e-06);
}

TempDep_ThermalConductivity::
~TempDep_ThermalConductivity ()
{
}

void TempDep_ThermalConductivity::
Compute_kappa (const double t, double& kappa)
{ kappa = 1.0 / (a + b*t + c*t*t);
}

void TempDep_ThermalConductivity::
Compute_dkappadt (const double t, double& dkappadt)
{ const double kappa = 1.0 / (a + b*t + c*t*t);
  dkappadt = -kappa * kappa * (b + 2.0*c*t);
}

extern "C"
```

```
PMI_ThermalConductivity* new_PMI_ThermalConductivity
  (const PMI_Environment& env, const PMI_AnisotropyType anisotype)
{ return new TempDep_ThermalConductivity (env, anisotype);
}
```

33.21 Heat capacity

The model for lattice heat capacity in (Eq. 15.25) can be specified in the `Physics` section of the DESSIS command file. The following two possibilities are available:

```
Physics {
   HeatCapacity (
      constant
      pmi_model_name
   )
}
```

These entries have the following meaning:

`constant`	Use constant heat capacity (default)
`pmi_model_name`	Call a PMI model to compute the heat capacity

33.21.1 Dependencies

The heat capacity c may depend on the variable:

`t`	Lattice temperature [K]

The PMI model must compute the following results:

`c`	Heat capacity c [JK^{-1}cm^{-3}]
`dcdt`	Derivative of c with respect to `t` [JK^{-2}cm^{-3}]

33.21.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_HeatCapacity : public PMI_Dessis_Interface {

public:
  PMI_HeatCapacity (const PMI_Environment& env);

  virtual ~PMI_HeatCapacity ();

  virtual void Compute_c
    (const double t, double& c) = 0;
  virtual void Compute_dcdt
    (const double t, double& dcdt) = 0

};
```

The following virtual constructor must be implemented:

```
typedef PMI_HeatCapacity* new_PMI_HeatCapacity_func
   (const PMI_Environment& env);
extern "C" new_PMI_HeatCapacity_func new_PMI_HeatCapacity;
```

33.21.3 Example: Constant heat capacity

The following C++ code implements constant heat capacity:

```
#include "PMIModels.h"

class Constant_HeatCapacity : public PMI_HeatCapacity {
private:
  double cv;

public:
  Constant_HeatCapacity (const PMI_Environment& env);
  ~Constant_HeatCapacity ();

  void Compute_c
    (const double t, double& c);

  void Compute_dcdt
    (const double t, double& dcdt);
};

Constant_HeatCapacity::
Constant_HeatCapacity (const PMI_Environment& env) :
  PMI_HeatCapacity (env)
{ // default values
  cv = InitParameter ("cv", 1.63);
}

Constant_HeatCapacity::
~Constant_HeatCapacity ()
{
}

void Constant_HeatCapacity::
Compute_c (const double t, double& c)
{ c = cv;
}

void Constant_HeatCapacity::
Compute_dcdt (const double t, double& dcdt)
{ dcdt = 0.0;
}

extern "C"
PMI_HeatCapacity* new_PMI_HeatCapacity
  (const PMI_Environment& env)
{ return new Constant_HeatCapacity (env);
}
```

33.22 Optical absorption

The optical absorption coefficient α in the photo generation can be accessed through a PMI. For each optical beam, a model can be specified in the `semabs` section of the DESSIS command file:

```
Physics {
   OptBeam (
      (...
      semabs (model = pmi_model_name)
      ...
      )
   )
}
```

33.22.1 Dependencies

The optical absorption coefficient α may depend on the following variables:

`energy`	Optical wave energy E_{ph} [eV]
`temperature`	Temperature T [K]

The PMI model must compute the following result:

`alpha`	Optical absorption coefficient α [cm^{-1}]

33.22.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Absorption : public PMI_Dessis_Interface {

public:
  PMI_Absorption (const PMI_Environment& env);
  virtual ~PMI_Absorption ();

  virtual void Compute_alpha
    (const double energy, const double t, double& alpha) = 0;
};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_Absorption* new_PMI_Absorption_func
   (const PMI_Environment& env);
extern "C" new_PMI_Absorption_func new_PMI_Absorption;
```

33.22.3 Example: Temperature-dependent absorption model

The following code computes a temperature-dependent value for the absorption coefficient α . This example is for illustration purposes only and does not pertain to any physical model:

```
#include "PMIModels.h"

class TempDep_Absorption : public PMI_Absorption {

private:
  double Alpha;

public:
  TempDep_Absorption (const PMI_Environment& env);

  ~TempDep_Absorption ();

  void Compute_alpha (const double energy, const double t, double& alpha);

};

TempDep_Absorption::
TempDep_Absorption (const PMI_Environment& env) :
  PMI_Absorption (env)
{ Alpha = InitParameter ("Alpha", 1e5);
}

TempDep_Absorption::
~TempDep_Absorption ()
{
}

void TempDep_Absorption::
Compute_alpha (const double energy, const double t, double& alpha)
{ alpha = Alpha*(600 - t)/300;
}

extern "C"
PMI_Absorption* new_PMI_Absorption
  (const PMI_Environment& env)
{ return new TempDep_Absorption (env);
}
```

33.23 Refractive index

The refractive index, n, in the photo generation can be accessed through a PMI. For each optical beam, a model can be specified in the `RefractiveIndex` section of the DESSIS command file:

```
Physics {
   RayTrace(
      ( ...
         RefractiveIndex(model = pmi_model_name)
         ...
      )
   )
}
```

33.23.1 Dependencies

The refractive index, n, may depend on the following variables:

`energy`	Optical wave energy E_{ph} [eV]
`temperature`	Temperature T [K]

The PMI model must compute the following result:

`Refract`	Refractive index n (1)

33.23.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_RefractiveIndex : public PMI_Dessis_Interface {

public:
  PMI_RefractiveIndex (const PMI_Environment& env);
  virtual ~PMI_RefractiveIndex ();

  virtual void Compute_Refract (const double energy, const double t, double& Refract) = 0;
};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_RefractiveIndex* new_PMI_RefractiveIndex_func
   (const PMI_Environment& env);
extern "C" new_PMI_RefractiveIndex_func new_PMI_RefractiveIndex;
```

33.23.3 Example: Temperature-dependent refractive index

The following code computes a simple temperature-dependent model for the refractive index:

```
#include "PMIModels.h"

class TempDep_RefractiveIndex : public PMI_RefractiveIndex {

private:
  double RIndex;

public:
  TempDep_RefractiveIndex (const PMI_Environment& env);
  ~TempDep_RefractiveIndex ();

  void Compute_Refract (const double energy, const double temp, double& out);
};

TempDep_RefractiveIndex::
TempDep_RefractiveIndex (const PMI_Environment& env) :
  PMI_RefractiveIndex (env)
{ RIndex = InitParameter ("RIndex", 42);
}

TempDep_RefractiveIndex::
~TempDep_RefractiveIndex ()
```

```
{
}

void TempDep_RefractiveIndex::
Compute_Refract (const double energy, const double temp, double& Refract)
{ Refract = 4*(1 + (temp-300)/100);
}

extern "C"
PMI_RefractiveIndex* new_PMI_RefractiveIndex (const PMI_Environment& env)
{ return new TempDep_RefractiveIndex (env);
}
```

33.24 Stress

DESSIS supports a PMI for mechanical stress (see Chapter 22 on page 15.353). The name of the PMI model must appear in the `Piezo` section of the command file:

```
Physics {
   Piezo (
      Stress = pmi_model_name
   )
}
```

33.24.1 Dependencies

A PMI stress model has no explicit dependencies. However, it can depend on doping concentrations and mole fractions through the run-time support.

The PMI model must compute the following results:

`stress_xx`	`XX` component of stress tensor [Pa]
`stress_yy`	`YY` component of stress tensor [Pa]
`stress_zz`	`ZZ` component of stress tensor [Pa]
`stress_yz`	`YZ` component of stress tensor [Pa]
`stress_xz`	`XZ` component of stress tensor [Pa]
`stress_xy`	`XY` component of stress tensor [Pa]

33.24.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Stress : public PMI_Dessis_Interface {

public:
  PMI_Stress (const PMI_Environment& env);
  virtual ~PMI_Stress ();

  virtual void Compute_StressXX
    (double& stress_xx) = 0;
  virtual void Compute_StressYY
    (double& stress_yy) = 0;

  virtual void Compute_StressZZ
    (double& stress_zz) = 0;

  virtual void Compute_StressYZ
    (double& stress_yz) = 0;

  virtual void Compute_StressXZ
    (double& stress_xz) = 0;

  virtual void Compute_StressXY
    (double& stress_xy) = 0;
};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_Stress* new_PMI_Stress_func
     (const PMI_Environment& env);
extern "C" new_PMI_Stress_func new_PMI_Stress;
```

33.24.3 Example: Constant stress model

The following code returns constant values for the stress tensor:

```
#include "PMIModels.h"

class Constant_Stress : public PMI_Stress {

private:
  double xx, yy, zz, yz, xz, xy;

public:
  Constant_Stress (const PMI_Environment& env);

  ~Constant_Stress ();

  void Compute_StressXX (double& stress_xx);
  void Compute_StressYY (double& stress_yy);
  void Compute_StressZZ (double& stress_zz);
  void Compute_StressYZ (double& stress_yz);
  void Compute_StressXZ (double& stress_xz);
  void Compute_StressXY (double& stress_xy);

};

Constant_Stress::
Constant_Stress (const PMI_Environment& env) :
  PMI_Stress (env)
{ xx = InitParameter ("xx", 100);
  yy = InitParameter ("yy", -4e9);
  zz = InitParameter ("zz", 300);
  yz = InitParameter ("yz", 400);
  xz = InitParameter ("xz", 500);
  xy = InitParameter ("xy", 600);
}

Constant_Stress::
~Constant_Stress ()
```

```
{
}

void Constant_Stress::
Compute_StressXX (double& stress_xx)
{ stress_xx = xx;
}

void Constant_Stress::
Compute_StressYY (double& stress_yy)
{ stress_yy = yy;
}

void Constant_Stress::
Compute_StressZZ (double& stress_zz)
{ stress_zz = zz;
}

void Constant_Stress::
Compute_StressYZ (double& stress_yz)
{ stress_yz = yz;
}

void Constant_Stress::
Compute_StressXZ (double& stress_xz)
{ stress_xz = xz;
}

void Constant_Stress::
Compute_StressXY (double& stress_xy)
{ stress_xy = xy;
}

extern "C"
PMI_Stress* new_PMI_Stress
  (const PMI_Environment& env)
{ return new Constant_Stress (env);
}
```

33.25 Trap space factor

The space distribution of traps can be computed by a PMI (see Chapter 10 on page 15.225). The name of the PMI is specified in the `Traps` section as follows:

```
Physics {
   Traps (sFactor=pmi_model_name  ...)
}
```

NOTE The name of the PMI model must not coincide with the name of an internal DESSIS field. Otherwise, DESSIS takes the value of the internal field as the space factor.

33.25.1 Dependencies

A PMI space factor model has no explicit dependencies. The model must compute:

`spacefactor`	Space factor (1)

33.25.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_SpaceFactor : public PMI_Dessis_Interface {

public:
  PMI_SpaceFactor (const PMI_Environment& env);
  virtual ~PMI_SpaceFactor ();

  virtual void Compute_spacefactor
    (double& spacefactor) = 0;
};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_SpaceFactor* new_PMI_SpaceFactor_func
    (const PMI_Environment& env);
extern "C" new_PMI_SpaceFactor_func new_PMI_SpaceFactor;
```

33.25.3 Example: PMI user field as space factor

The following code reads the space factor from a PMI user field:

```
#include "PMIModels.h"

class pmi_spacefactor : public PMI_SpaceFactor {

public:
  pmi_spacefactor (const PMI_Environment& env);
  ~pmi_spacefactor ();

  void Compute_spacefactor (double& spacefactor);
};

pmi_spacefactor::
pmi_spacefactor (const PMI_Environment& env) :
  PMI_SpaceFactor (env)
{
}

pmi_spacefactor::
~pmi_spacefactor ()
{
}

void pmi_spacefactor::
Compute_spacefactor (double& spacefactor)
{ spacefactor = ReadUserField (PMI_UserField1);
}
```

```
extern "C"
PMI_SpaceFactor* new_PMI_SpaceFactor
  (const PMI_Environment& env)
{ return new pmi_spacefactor (env);
}
```

33.26 Piezoelectric polarization

The effects of piezoelectric polarization can be modeled by adding an additional charge term:

$$q_{\mathrm{PE}} = \mathrm{div} P_{\mathrm{PE}} \tag{15.675}$$

to the right-hand side of the Poisson equation (see (Eq. 15.19)). The quantity P_{PE} denotes the piezoelectric polarization vector, which may defined by a PMI.

The name of the PMI is specified in the `Physics` section of the DESSIS command file as follows:

```
Physics {
    Piezoelectric_Polarization (pmi_polarization)
}
```

The piezoelectric polarization vector and the piezoelectric charge may be plotted by:

```
Plot {
   PE_Polarization/vector
   PE_Charge
}
```

DESSIS assumes that the piezoelectric polarization vector P_{PE} is zero outside of the device. This boundary condition may lead to an unexpectedly large charge density if P_{PE} has a nonzero component orthogonal to the boundary (discontinuity in $\mathrm{div} P_{\mathrm{PE}}$).

33.26.1 Dependencies

The piezoelectric polarization model does not have explicit dependencies. However, it can use the run-time support. In particular, it has access to the stress fields.

The model must compute:

`pol`	Piezoelectric polarization vector [Ccm^{-2}]

The resulting vector `pol` has dimension 3. However, only the first *dim* components need to be defined, where *dim* is equal to the dimension of the problem.

33.26.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_Polarization : public PMI_Dessis_Interface {

public:
  PMI_Polarization (const PMI_Environment& env);
  virtual ~PMI_Polarization ();

  virtual void Compute_pol
    (double pol [3]) = 0;
};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_Polarization* new_PMI_Polarization_func
    (const PMI_Environment& env);
extern "C" new_PMI_Polarization_func new_PMI_Polarization;
```

33.26.3 Example: Gaussian polarization model

In this example, the piezoelectric polarization vector P_{PE} has a simple Gaussian shape in the x-direction:

```
#include "PMIModels.h"

class Gauss_Polarization : public PMI_Polarization {

private:
  double x0, c, a;

public:
  Gauss_Polarization (const PMI_Environment& env);
  ~Gauss_Polarization ();

  void Compute_pol (double pol [3]);
};


Gauss_Polarization::
Gauss_Polarization (const PMI_Environment& env) :
  PMI_Polarization (env)
{ x0 = InitParameter ("x0", 0.0);
  c = InitParameter ("c", 1.0);
  a = InitParameter ("a", 1e-5);
}

Gauss_Polarization::
~Gauss_Polarization ()
{
}

void Gauss_Polarization::
Compute_pol (double pol [3])
{ double x, y, z;
  ReadCoordinate (x, y, z);
  pol [0] = a * exp (-c * (x-x0) * (x-x0));
  pol [1] = 0.0;
  pol [2] = 0.0;
}
```

```
extern "C"
PMI_Polarization* new_PMI_Polarization
  (const PMI_Environment& env)
{ return new Gauss_Polarization (env);
}
```

33.27 Incomplete ionization

The ionization factors $G_D(T)$ and $G_A(T)$ (see Section 6.3 on page 15.162) can be defined by a PMI.

The name of the PMI should be specified in the `Physics` section of the DESSIS command file as follows:

```
Physics {
   IncompleteIonization( Model( PMI_model_name("Species_name1 Species_name2 ...") ) )
}
```

In addition, it is possible to have a PMI for each species separately:

```
Physics {
   IncompleteIonization(
      Model(
         PMI_model_name1("Species_name1")
         PMI_model_name2("Species_name2")
      )
   )
}
```

The species PMI parameters should be defined in the DESSIS parameter file (see Section 33.6 on page 15.542).

33.27.1 Dependencies

The ionization factors $G_D(T)$ and $G_A(T)$ may depend on the variable:

`T`	Lattice temperature [K]

The PMI model must compute the following results:

`g`	Ionization factor $G(T)$
`dgdt`	Derivative of $G(T)$ with respect to `T`

33.27.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
enum PMI_SpeciesType {
  PMI_acceptor,
  PMI_donor
};

class PMI_DistributionFunction : public PMI_Dessis_Interface {
```

```
private:
  const PMI_SpeciesType speciesType;
  const char* speciesName;

public:
  PMI_DistributionFunction (const PMI_Environment& env,
                            const PMI_SpeciesType type,
                            const char* name);

  virtual ~PMI_DistributionFunction ();

  PMI_SpeciesType SpeciesType () const { return speciesType; }
  const char* SpeciesName () const { return speciesName; }

// read parameter from Dessis parameter file
// (override for PMI_Dessis_Interface::ReadParameter)
  const PMIBaseParam* ReadParameter (const char* name) const;

// initialize parameter from Dessis parameter file or from default value
// (override for PMI_Dessis_Interface::InitParameter)
  double InitParameter (const char* name, double defaultvalue) const;

virtual void Compute_g
    (const double T,        // lattice temperature
     double& g) = 0;        // g = G(T)

  virtual void Compute_dgdt
    (const double T,        // lattice temperature
     double& dgdt) = 0;     // dgdt = G'(T)

};
```

The prototype for the virtual constructor is given as:

```
typedef PMI_DistributionFunction* new_PMI_DistributionFunction_func
    (const PMI_Environment& env);
extern "C" new_PMI_DistributionFunction_func new_PMI_DistributionFunction;
```

33.27.3 Example: Matsuura incomplete ionization model

The following C++ code implements the Matsuura model [179] for dopant Al in SiC material:

$$G_A(T) = 4 \cdot \exp\left(\frac{\Delta E_A - E_{ex}}{k_B T}\right) \cdot \left(g_1 + \sum_{r=2} g_r \exp\left(\frac{\Delta E_r - \Delta E_A}{k_B T}\right)\right) \tag{15.676}$$

where g_1 is the ground-state degeneracy factor, g_r is the $(r-1)$-th excited state degeneracy factor, and ΔE_r is the difference in energy between the $(r-1)$-th excited state level and E_V. ΔE_r is given by the hydrogenic dopant model [180]:

$$\Delta E_r = 13.6 \cdot \frac{m^*}{m_0 \cdot \varepsilon_s^2} \cdot \frac{1}{r^2} \qquad \text{[eV]} \tag{15.677}$$

where m_0 is the free-space electron mass, m^* is the hole effective mass in SiC, and ε_s is the dielectric constant of SiC. The acceptor level is described as [180]:

$$\Delta E_A = \Delta E_1 + E_{CCC} \tag{15.678}$$

where E_{CCC} is the energy induced due to central cell corrections.

The ensemble average E_{ex} of the ground and excited state levels of the acceptor is given by [181]:

$$E_{ex} = \frac{\sum_{r=2} (\Delta E_A - \Delta E_r) g_r \exp\left(-\frac{\Delta E_A - \Delta E_r}{k_B T}\right)}{g_1 + \sum_{r=2} g_r \exp\left(-\frac{\Delta E_A - \Delta E_r}{k_B T}\right)} \tag{15.679}$$

The Matsuura model can be implemented as follows:

```
class Matsuura_DistributionFunction : public PMI_DistributionFunction {

protected:
  const double kB_300;  //  Boltzmann constant * 300   [eV]
  int   nb_item;        //  number of item in sum
  double *gr, *dEr;
  double Eex, dEA, Eccc;

public:
  Matsuura_DistributionFunction (const PMI_Environment& env,
                                 const char* name,
                                 const PMI_SpeciesType type = PMI_acceptor);

  ~Matsuura_DistributionFunction ();

  void Compute_g
    (const double T,        // lattice temperature
     double& g);            // g = G(T)


  void Compute_dgdt
    (const double T,        // lattice temperature
     double& dgdt);         // dgdt = G'(T)

   double Compute_Eex(double T);   // compute Eex(T) [15.679]
   double Compute_dEexdT(double T);// compute dEex/dT

};

Matsuura_DistributionFunction::
Matsuura_DistributionFunction (const PMI_Environment& env,
                               const char* name,
                               const PMI_SpeciesType type) :
  PMI_DistributionFunction (env, name, type),
  kB_300(1.380662e-23*300./1.602192e-19),  //  kB*T0/e0 = 0.02585199527 [eV]
  Eex(0.)
{
  nb_item  = InitParameter ("NumberOfItem", 1);
  Eccc     = InitParameter ("Eccc", 0);

  if(nb_item < 1) {
    printf("ERROR; PMI model Matsuura_DistributionFunction: parameter NumberOfItem < 1 \n");
    exit(1);
  }
  gr  = new double[nb_item];
  dEr = new double[nb_item];

  char str_r[6], name_gr[6];
  int r;
  for(r=0; r<nb_item; ++r) {
    name_gr[0] = 'g';  name_gr[1] = '\0';
    sprintf(str_r, "%d\0", r+1);
    strcat(name_gr, str_r);
    const PMIBaseParam* par = ReadParameter(name_gr);
    if(!par) {
      printf("ERROR; PMI model Matsuura_DistributionFunction: cannot read parameter %s \n", name_gr);
     gr[r] = 2;
} else
      gr[r] = *par;
  }

  const PMIBaseParam* par = ReadParameter("dE1");
  if(!par) {
   //  dE[r] = 13.6 * m_eff/m0/eps/eps/r/r
    Compute_dEr(nb_item, dEr);
  } else {
    char name_dEr[6];
    for(r=0; r<nb_item; ++r) {
      strcpy(name_dEr, "dE");
      sprintf(str_r, "%d\0", r+1);
      strcat(name_dEr, str_r);
      const PMIBaseParam* par = ReadParameter(name_dEr);
      if(!par) {
        printf("ERROR; PMI model Matsuura_DistributionFunction: cannot read parameter %s \n", name_dEr);
         exit(1);
      }
      dEr[r] = *par;
    }
  }
  dEA = dEr[0] + Eccc;
}


 Matsuura_DistributionFunction::
~Matsuura_DistributionFunction ()
{
  delete[] gr;
  delete[] dEr;
}

void Matsuura_DistributionFunction::Compute_g
    (const double T,                 // lattice temperature
     double& g)                      // g = G(T)
{
  const double kT = kB_300*T/300;

  Eex = Compute_Eex(T);
  g = gr[0];

  for(int r=1; r<nb_item; ++r) {
    double delta = dEA - dEr[r];
    g += gr[r]*exp( -delta/kT );
  }
  g *= 4.*exp( (dEA-Eex)/kT );
}

void Matsuura_DistributionFunction::Compute_dgdt
    (const double T,       // lattice temperature
     double& dgdt)         // dgdt = G'(T)
{
  const double kT = kB_300*T/300;
  Eex = Compute_Eex(T);
  double s1, s2 = gr[0], s3, s4 = 0., delta, tmp;

  for(int r=1; r<nb_item; ++r) {
    delta = dEA - dEr[r];
    tmp   = gr[r]*exp( -delta/kT );
    s2    += tmp;
    s4    += tmp*delta/kT/T;                //  s4 = ds2/dT
  }

  delta = dEA - Eex;
  s1 = 4.*exp( delta/kT );
  double dEex_dT = Compute_dEexdT(T);
  s3 = s1*( -dEex_dT/kT - delta/kT/T );   //  s3 = ds1/dT

  dgdt = s3*s2 + s1*s4;                //  dgdt = d(s1*s2)/dT

  return;
}

//  Eex is given by [15.679]
double Matsuura_DistributionFunction::Compute_Eex(double T)
{
  const double kT = kB_300*T/300;

  double s1 = 0., s2 = gr[0];

  for(int r=1; r<nb_item; ++r) {
    double delta = dEA - dEr[r];
    double tmp   = gr[r]*exp( -delta/kT );
    s1 += delta*tmp;
    s2 += tmp;
  }

  return s1/s2;

}

double Matsuura_DistributionFunction::Compute_dEexdT(double T)
{
  const double kT = kB_300*T/300;

  double s1 = 0., s2 = gr[0], s3 = 0., s4 = 0.;

  for(int r=1; r<nb_item; ++r) {
    double delta = dEA - dEr[r];
    double tmp   = gr[r]*exp( -delta/kT );
    s1 += delta*tmp;
    s2 += tmp;
    s3 += delta*tmp*delta/kT/T;  //  s3 = ds1/dT
    s4 += tmp*delta/kT/T;        //  s4 = ds2/dT
  }

  return (s3*s2 - s1*s4)/s2/s2;

}


extern "C"
PMI_DistributionFunction* new_PMI_DistributionFunction
  (const PMI_Environment& env,
   const char* name,
   const PMI_SpeciesType type)
{
  return new  Matsuura_DistributionFunction (env, name, type);
}


void Compute_dEr(int nb_item, double* dEr)
{
  // dEr is given by [15.677]

  //  data from file: 6H-SiC.par
  const double epsilon = 9.66;                     //  dielectric constant
  const double mh      = 1;                        //  hole effective mass in SiC

  const double E0 = 13.6*mh/epsilon/epsilon;

  for(int r=1; r<=nb_item; ++r) {
    dEr[r-1] = E0/r/r;
  }
}
```

33.28 Current plot

The current plot PMI allows user-computed entries to be added to the DESSIS current plot file. It is specified in the `CurrentPlot` section of the command file, for example:

```
CurrentPlot {
   pmi_CurrentPlot
}
```

The interface has access to the device mesh and device data.

33.28.1 Structure of current plot file

A DESSIS current plot file consists of a header section and a data section. For each function, the structure can be described as follows:

```
dataset name
function name
value$_0$
value$_1$
...
```

A dataset name denotes a dataset, for example:

```
time
Tmin
```

If a dataset corresponds to a region or contact, it is customary to add the region or contact name:

```
gate Charge
```

The function name describes the function, for example:

```
ElectrostaticPotential
Temperature
```

Afterwards, a function value is added to the current plot file for each plot time point.

33.28.2 C++ interface

The following base class is declared in the file `PMIModels.h`:

```
class PMI_CurrentPlot : public PMI_Dessis_Device_Interface {

public:
  PMI_CurrentPlot (const PMI_Device_Environment& env);
  virtual ~PMI_CurrentPlot ();

  virtual void Compute_Dataset_Names
    (des_string_vector& dataset) = 0;

  virtual void Compute_Function_Names
    (des_string_vector& function) = 0;

  virtual void Compute_Plot_Values
    (des_double_vector& value) = 0;
};
```

The methods `Compute_Dataset_Names()` and `Compute_Function_Names()` are used to generate the header in the DESSIS current plot file (see Section 33.28.1 on page 15.600). `Compute_Plot_Values()` is called for each plot time point to compute the plot values. Use the `push_back()` function to add values to the arrays `dataset`, `function`, or `value`.

NOTE All three methods `Compute_Dataset_Names()`, `Compute_Function_Names()`, and `Compute_Plot_Values()` must always compute the same number of values. Otherwise, an inconsistent current plot file will be generated.

The prototype for the virtual constructor is given as:

```
typedef PMI_CurrentPlot* new_PMI_CurrentPlot_func
   (const PMI_Device_Environment& env);
extern "C" new_PMI_CurrentPlot_func new_PMI_CurrentPlot;
```

33.28.3 Run-time support

The class `PMI_Dessis_Device_Interface` provides run-time support:

```
class PMI_Dessis_Device_Interface {

public:
  PMI_Dessis_Device_Interface (const PMI_Device_Environment& env);
  virtual ~PMI_Dessis_Device_Interface ();

  const char* Name () const;

  const PMIBaseParam* ReadParameter (const char* name) const;
```

```
    double InitParameter (const char* name, double defaultvalue) const;

    double ReadTime () const;

    const des_mesh* Mesh () const;
    des_data* Data () const;
  };
```

The method `Name()` returns the name of the PMI model as specified in the DESSIS command file. The methods `ReadParameter()` and `InitParameter()` read the value of a parameter from the DESSIS parameter file (see Section 33.6 on page 15.542).

NOTE Parameters for the current plot PMI must appear in the global parameter section. Regionwise or materialwise parameters are not supported.

`ReadTime()` returns the simulation time during a transient simulation [s]. The methods `Mesh()` and `Data()` provide access to the DESSIS mesh and data (see Section 33.28.4 and Section 33.28.5 on page 15.606).

33.28.4 Device mesh

A DESSIS device mesh consists of a number of regions. A region is either a contact region consisting of a list of contact vertices or a bulk region consisting of a list of elements. An element is described by a list of vertices.

33.28.4.1 Vertex

In the file `PMIModels.h`, the class `des_vertex` is declared as follows:

```
  class des_vertex {

  public:
    size_t index () const;

    const double* coord () const;

    bool equal_coord (des_vertex* v) const;

    size_t size_edge () const;
    des_edge* edge (size_t i) const;

    size_t size_element () const;
    des_element* element (size_t i) const;

    size_t size_region () const;
    des_region* region (size_t i) const;
  };
```

The value of `index()` can be used as an index for vertex-based data (see Section 33.28.5).

The location of a vertex is given by its coordinates `coord()`. The function `equal_coord()` should be used to check if two vertices have the same coordinates. For example, DESSIS duplicates vertices along heterointerfaces. Consequently, two vertices with different indices can share the same coordinates.

`size_edge()` returns the number of edges connected to a vertex. The method `edge()` can be used to retrieve the ith edge.

`size_region()` returns the number of regions containing a vertex. The method `region()` can be used to retrieve the ith region.

33.28.4.2 Edge

In the file `PMIModels.h`, the class `des_edge` is declared as follows:

```
class des_edge {

public:
  size_t index () const;

  des_vertex* start () const;
  des_vertex* end () const;

  size_t size_element () const;
  des_element* element (size_t i) const;

  size_t size_region () const;
  des_region* region (size_t i) const;
};
```

The value of `index()` can be used as an index for edge-based data (see Section 33.28.5 on page 15.606).

`start()` and `end()` return the first and second vertex connected to the edge, respectively.

`size_element()` returns the number of elements connected to an edge. The method `element()` can be used to retrieve the ith element.

`size_region()` returns the number of regions containing an edge. The method `region()` can be used to retrieve the ith region.

33.28.4.3 Element

In the file `PMIModels.h`, the class `des_element` is declared as follows:

```
class des_element {

public:
  typedef enum { point, line, triangle, rectangle, tetrahedron,
                 pyramid, prism, cuboid, tetrabrick } des_type;

  size_t index () const;

  des_type type () const;

  size_t size_vertex () const;
  des_vertex* vertex (size_t i) const;
```

```
    size_t size_edge () const;
    des_edge* edge (size_t i) const;

    des_bulk* bulk () const;
  };
```

The value of `index()` can be used as an index for element-based data (see Section 33.28.5 on page 15.606).

`type()` returns the type of an element (point, line, triangle, rectangle, tetrahedron, pyramid, prism, cuboid, or tetrabrick). An element is mainly described by its vertices. `size_vertex()` returns the number of vertices in an element, and the method `vertex()` can be used to retrieve the ith vertex. `size_edge()` returns the number of edges of an element. The method `edge()` can be used to retrieve the ith edge. The method `bulk()` returns the bulk region containing the element.

Figure 15.122 shows the numbering of vertices and edges for all element types.

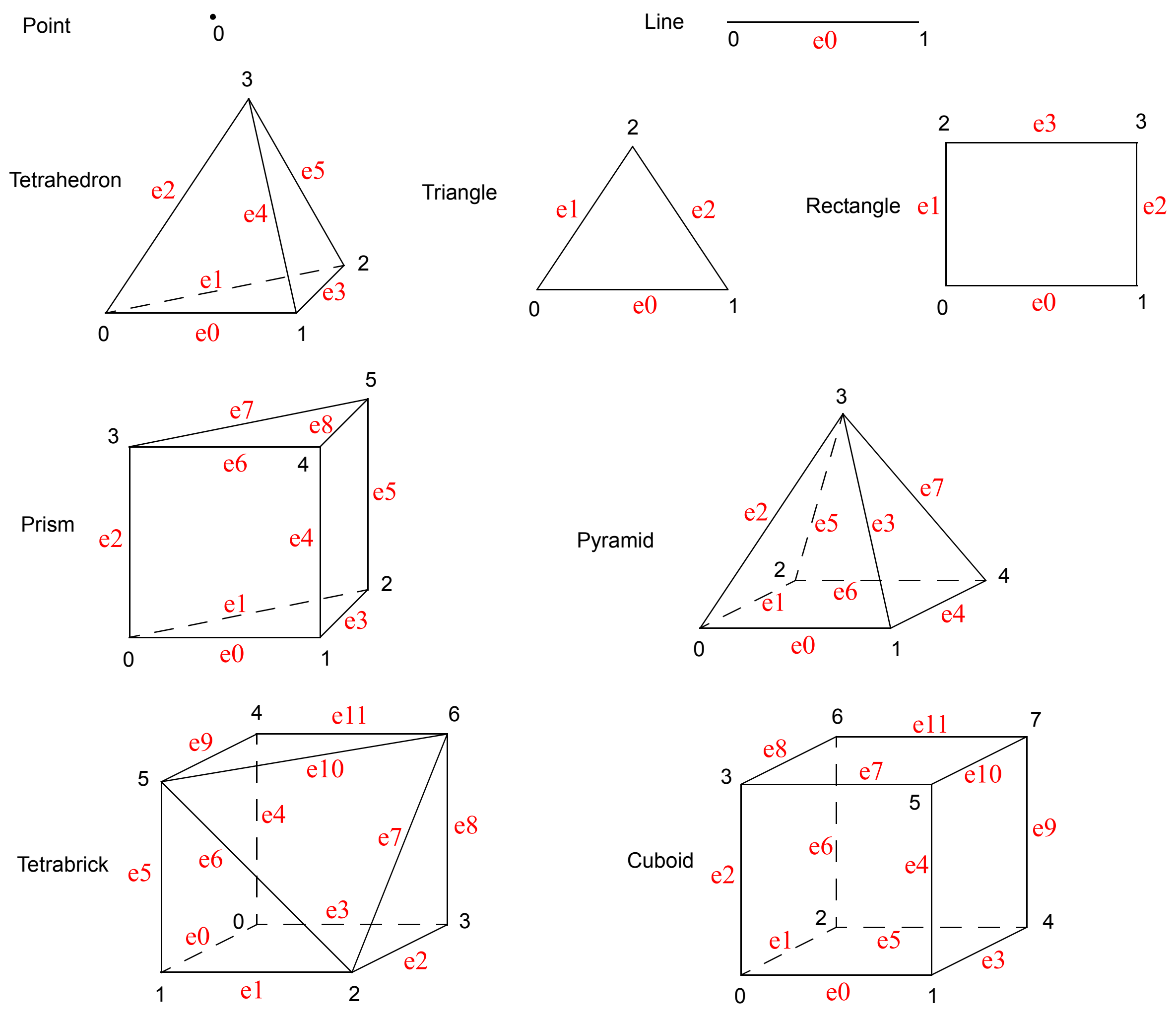


Figure 15.122   Vertex and edge numbering

33.28.4.4 Region

In the file `PMIModels.h`, the base class `des_region` is declared as follows:

```
class des_region {

public:
  typedef enum { bulk, contact } des_type;

  virtual des_type type () const = 0;

  std::string name () const;

  size_t size_vertex () const;
  des_vertex* vertex (size_t i) const;

  size_t size_edge () const;
  des_edge* edge (size_t i) const;
};
```

A DESSIS mesh consists of two types of regions, bulk regions and contacts. The virtual method `type()` returns the type of a region. The name of a region is returned by `name()`. `size_vertex()` returns the number of vertices in a region. The method `vertex()` can be used to retrieve the ith vertex. `size_edge()` returns the number of edges in a region. The method `edge()` can be used to retrieve the ith edge.

The class `des_bulk` is derived from `des_region`:

```
class des_bulk : public des_region {

public:
  des_type type () const;

  std::string material () const;

  size_t size_element () const;
  des_element* element (size_t i) const;
};
```

`material()` returns the name of the material in a bulk region. `size_element()` returns the number of elements in a region. The method `element()` can be used to retrieve the ith element.

Similarly, the class `des_contact` is also derived from `des_region`:

```
class des_contact : public des_region {

public:
  des_type type () const;
};
```

33.28.4.5 Mesh

In the file `PMIModels.h`, the class `des_mesh` is declared as follows:

```
class des_mesh {

public:
  int dim () const;
```

```
    size_t size_vertex () const;
    des_vertex* vertex (size_t i) const;

    size_t size_edge () const;
    des_edge* edge (size_t i) const;

    size_t size_element () const;
    des_element* element (size_t i) const;

    size_t size_region () const;
    des_region* region (size_t i) const;
  };
```

The dimension of the mesh is given by `dim()`. The possible values are 1, 2, and 3.

`size_vertex()` returns the number of vertices in the mesh. The method `vertex()` can be used to retrieve the ith vertex. `size_edge()` returns the number of edges in the mesh. The method `edge()` can be used to retrieve the ith edge. `size_element()` returns the number of elements in the mesh. The method `element()` can be used to retrieve the ith element. `size_region()` returns the number of regions in the mesh. The method `region()` can be used to retrieve the ith region.

33.28.5 Device data

In the file `PMIModels.h`, the class `des_data` is declared as follows:

```
class des_data {

public:
  typedef enum { vertex, edge, element } des_location;

  const double*const* ReadCoefficient ();
  const double*const* ReadMeasure ();

  const double* ReadScalar (des_location location, std::string name);
  const double*const* ReadVector (des_location location, std::string name);
};
```

The methods `ReadCoefficient()` and `ReadMeasure()` return the box-method coefficients κ_{ij} and measure μ_{ij} used in DESSIS (see Section 32.1 on page 15.519).

`ReadCoefficient()` returns a two-dimensional array. The two indices are the element index and the local edge number.

The following code fragment reads the coefficients for all element edges:

```
const des_mesh* mesh = Mesh();
des_data* data = Data();
const double*const* coeff = data->ReadCoefficient();
for (size_t eli = 0; eli < mesh->size_element(); eli++) {
  des_element* el = mesh->element(eli);
  for (size_t ei = 0; ei < el->size_edge(); ei++) {
    des_edge* e = el->edge(ei);
    const double c = coeff[el->index()][ei];
  }
}
```

NOTE The values κ_{ij} returned by `ReadCoefficient()` are element-edge coefficients. The edge coefficients κ_i can be obtained by adding the contributions from all elements connected to an edge i.

`ReadMeasure()` returns a two-dimensional array. The two indices are the element index and the local vertex number. The following code fragment reads the measures for all element vertices:

```
const des_mesh* mesh = Mesh();
des_data* data = Data();
const double*const* measure = data->ReadMeasure();
for (size_t eli = 0; eli < mesh->size_element(); eli++) {
  des_element* el = mesh->element(eli);
  for (size_t vi = 0; vi < el->size_vertex(); vi++) {
    des_vertex* v = el->vertex(vi);
    const double m = measure[el->index()][vi];
  }
}
```

NOTE The values μ_{ij} returned by `ReadMeasure()` are element-vertex measures. The node measures μ_i can be obtained by adding the contributions from all elements connected to a vertex i.

The methods `ReadScalar()` and `ReadVector()` provide access to the DESSIS data. The values can be located on vertices, elements, or edges. See Table 15.167 on page 15.619 and Table 15.168 on page 15.627 for an overview of available scalar and vector data.

`ReadScalar()` returns a one-dimensional array. Use the `index()` method in the classes `des_vertex`, `des_edge`, or `des_element` to access the array elements.

`ReadVector()` returns a two-dimensional array. The first index selects the dimension (0, 1, 2) and the second index is used in the same way as for scalar data.

33.28.6 Example: Average electrostatic potential

The following example computes regionwise averages for the electrostatic potential. This is the same functionality as provided by the built-in current plot command (see Section 2.7 on page 15.52):

```
class CurrentPlot : public PMI_CurrentPlot {
private:
  typedef std::vector<des_bulk*> des_bulk_vector;

  const des_mesh* mesh;     // device mesh
  des_bulk_vector regions;  // list of semiconductor bulk regions
  double scale;             // scaling factor

public:
  CurrentPlot (const PMI_Device_Environment& env);
  ~CurrentPlot ();

  void Compute_Dataset_Names (des_string_vector& dataset);
  void Compute_Function_Names (des_string_vector& function);
  void Compute_Plot_Values (des_double_vector& value);
};
```

```
CurrentPlot::
CurrentPlot (const PMI_Device_Environment& env) :
  PMI_CurrentPlot (env)
{ mesh = Mesh ();
  // determine regions to process
  for (size_t ri = 0; ri < mesh->size_region (); ri++) {
    des_region* r = mesh->region (ri);
    if (r->type () == des_region::bulk) {
      des_bulk* b = dynamic_cast <des_bulk*> (r);
      if (b->material () != "Oxide") {
        // we found a semiconductor bulk region
        regions.push_back (b);
      }
    }
  }

  // read parameters
  scale = InitParameter ("scale", 0.0);
}

CurrentPlot::
~CurrentPlot ()
{
}

void CurrentPlot::
Compute_Dataset_Names (des_string_vector& dataset)
{ for (size_t ri = 0; ri < regions.size (); ri++) {
    des_bulk* b = regions [ri];
    std::string name = "Average_";
    name += b->name ();
    name += "ElectrostaticPotential";
    dataset.push_back (name);
  }
}

void CurrentPlot::
Compute_Function_Names (des_string_vector& function)
{ for (size_t ri = 0; ri < regions.size (); ri++) {
    function.push_back ("ElectrostaticPotential");
  }
}

void CurrentPlot::
Compute_Plot_Values (des_double_vector& value)
{ des_data* data = Data ();
  const double*const* measure = data->ReadMeasure ();
  const double* pot = data->ReadScalar (des_data::vertex, "ElectrostaticPotential");
  for (size_t ri = 0; ri < regions.size (); ri++) {
    des_bulk* b = regions [ri];

    double sum_pot = 0.0;
    double sum_measure = 0.0;

    for (size_t ei = 0; ei < b->size_element (); ei++) {
      des_element* e = b->element (ei);
      for (size_t vi = 0; vi < e->size_vertex (); vi++) {
        des_vertex* v = e->vertex (vi);
        const double m = measure [e->index ()][vi];
        const double p = pot [v->index ()];
        sum_pot += m * p;
        sum_measure += m;
      }
    }

    value.push_back (scale * (sum_pot / sum_measure));
  }
}

extern "C" {
PMI_CurrentPlot* new_PMI_CurrentPlot (const PMI_Device_Environment& env)
{ return new CurrentPlot (env);
}
}
```

APPENDIX A Syntax

The syntax of the DESSIS input file, and the basic syntactical and lexical conventions are described here. DESSIS has a hierarchical input syntax. At the lowest level, `device`, `system`, and `solve` information is specified as well as the default and global parameters. Inside each `Dessis` section, the parameters specific to one device type can be specified.

Inside the `System` section, the real devices are specified or 'instantiated.' Here, parameters can be given that are specific to one instantiation of a device. The DESSIS input file is a collection of specifications used to establish the simulation environment with actions describing which equations must be solved and how they must be solved. The syntax of the DESSIS input file contains several entry types. All basic input file entries adhere to the syntactical and lexical rules described in Table 15.164.

Table 15.164 Entry types in DESSIS

Entry type	Description
Keyword	These are the known names of the input file. They are case insensitive. Therefore, the following keywords are all equivalent: `Quasistationary`, `QuasiStationary`, and `quasistationary`. Most keywords can be abbreviated. The above example can also be written as `QuasiStat`.
Integer	These are (possibly) signed decimal numbers. The following integers are valid: `123`, `-73492`, `0`.
Float	Floating point numbers are compatible with the C language format for floating point numbers. The following floating point numbers are valid: `123`, `123.0`, `1.23e2`, `-1.23E2`.
Vector	Vectors in real space are defined depending on the actual dimension. In 3D, a vector is specified by three floating point numbers; in 2D, by two floating point numbers enclosed in parentheses. The floats are separated by commas or spaces. In 1D, one floating point number without parentheses is sufficient. Valid vectors are `(1,0,2)`, `(1e-4,-1e-3)`, and `1`.
String	Strings are delimited by quotation marks. They are compatible with the C language format for strings. The following strings are valid: `"Vdd"`, `"output/diode"`.
Identifier	These are used to name objects such as nodes, devices, or attributes. They are compatible with the C language format for identifiers. The following identifiers are valid: `Vdd`, `diode`, `bjt_345`.
Assignment	These are used to set values to keywords. Therefore, the following are valid assignments: `Digits=4`, `Save="output/diode"`.
Signal	Signals are time dependent, piecewise, linear functions (not to be confused with UNIX signals) that are defined as inputs on the contacts of a device. They are specified as follows: (value0 at time0, value1 at time1, ... value_n at time_n). The following signal is valid: `(0 at 0, 1 at 10.0e-9, 1 at 20.0e-9)`.

List	Lists are collections of keywords, assignments, and complex entries. They are delimited by "(...)" or "{...}". The following lists are valid:
	`{ Number=0 Voltage=0`
	` Voltage=( 0 at 0, 0 at 2e-8 ) }`
	`{ Method=Super Digits=6 Numerically }`
	`( MinStep=1e-15 InitialStep=1e-10 Digits=3 )`

Structured entries	These are parameterized definitions or commands that can have the forms: `<keyword> {<keywords>}`, `<keyword> (<keywords>)`, or `<keyword> (<list>) {<list>}`.

APPENDIX B File-naming convention

The data exchange format DF-ISE defines the file-naming convention for ISE tools. The relevant items for DESSIS are described here.

All strings that represent file names containing a dot (.) within their base name are taken literally. Otherwise, DESSIS extends the given strings with the appropriate extension.

DESSIS expands the extensions for output files by its tool extension `_des`, for example, the extension of a saved file is `_des.sav`.

Compressed files are additionally extended by the extension `.Z`, for example, compressed plot files have the extension `_des.dat.Z`. Table 15.165 summarizes the extensions used in DESSIS.

Table 15.165 DESSIS file-naming convention

File	I/O	Extension
Command	I	`_des.cmd`, `.cmd`
Log	O	`_des.log`
Parameter	I	`.par`
Geometry	I	`.grd`
Doping	I	`.dat`
Lifetime	I	`.dat`
Save	O	`_des.sav`
Load	I	`.sav`
Device Plot (grid-based)	O	`_des.dat`
Current Plot	O	`_des.plt`
AC Extraction	O	`_ac_des.plt`
Montecarlo	I/O	See SPARTA manual.

During transient simulations, quasistationaries, and continuations, the plot and save files are numbered by a global index.

B.1 Compatibility with old file-naming convention

To be compatible with the old file-naming conventions, DESSIS tries to find the files according to the current convention and, if the corresponding files are not found, it tries to read the files according to the old convention. Table 15.166 summarizes the old convention.

Table 15.166 Old file-naming convention

File	I/O	Extension
Command	I	`.in`
Output or Log	O	`.out`
Parameter	I	`.par`
Geometry	I	`.geo`
Doping	I	`.dop`
Lifetime	I	`.life`
Save	O	`.sav`
Load	I	`.sav`
Device Plot (grid-based)	O	`.prt`
Current Plot	O	`.cur`
AC Extraction	O	`.ac`

APPENDIX C Command-line options

To start DESSIS, enter:

```
dessis [<options>] [<commandfile>]
```

DESSIS appends automatically the corresponding extension to the given command file if necessary. If no command file is specified, DESSIS reads from standard input.

DESSIS interprets the following options:

`-d`	Prints debug information into the `debug` file. The information printed includes the numeric values of the Jacobian and RHS for each equation at each solution step.
`-h`	Lists these options and exits.
`-i`	Prints the initial solution in the save file, and print files specified in the `File` section of the command file, and exits without performing further computations.
`-n`	Does not include Newton information in the `log` file.
`-P`	Writes the silicon model parameters into a file `dessis.par` and exits. This file can be modified and reloaded into DESSIS to make customized changes to physical models and parameters.
`-P:<Material>`	Writes the model parameters for the given material into a file `dessis.par` and exits.
`-P:All`	Writes the model parameters for all materials into a file `dessis.par` and exits.
`-P:<Material>/<Material>`	Writes the model parameters for the given material interface into a file `dessis.par` and exits.
`-P <commandfile>`	Writes the model parameters for the materials and interfaces used in `<commandfile>` into a file `dessis.par` and exits.
`-L`	Writes the silicon model parameters into the file `Silicon.par` and exits.
`-L:<Material>`	Writes a model parameter file `<Material>.par` for the specified material and exits.
`-L:All`	Writes a separate model parameter file for all materials and exits.
`-L:<Material>/<Material>`	Writes a model parameter file `<Material>%<Material>.par` for the specified material interface and exits.
`-L <commandfile>`	Writes model parameter files for all the materials, material interfaces, and electrodes used in `<commandfile>` and exits.
`-q`	Quiet mode for output.
`-S`	Writes the `SiC model` parameters into a `dessisSiC.par` file and exits. This file can be modified and reloaded into DESSIS to make customized changes to physical models and parameters.

`-v`	Prints header with version number of DESSIS.
`-V`	Prints detailed version information about DESSIS.
`--parameter-names`	Prints the names of the parameters from the DESSIS parameter file that can be ramped. If a command file is also supplied, DESSIS prints the parameters from the command file that can be ramped.
`--exit-on-failure`	Terminates immediately after a failed solve command.
`--compiler-version`	Prints the version of the C++ compiler that was used to compile DESSIS.

In addition, generic tool options can be found in the relevant section of the *ISE TCAD Release Notes*.

APPENDIX D Run-time statistics

The command `dessisstat` displays some statistics of a previous run of DESSIS based on the information found in its `log` file. For example, the command:

```
dessisstat test_des.log
```

generates the following statistics:

```
Total number of Newton iterations : 5
Total CPU-time                    : 3.3 s
Rhs-time                          : 19.39 % ( 0.64 s )
Jacobian-time                     : 45.15 % ( 1.49 s )
Solve-time                        : 32.42 % ( 1.07 s )
```

APPENDIX E Data and plot names

Table 15.167 and Table 15.168 on page 15.627 list the plot names that are recognized in a DESSIS `Plot` section (see Section 2.6 on page 15.52) and the data names that are available in the current plot PMI (see Section 33.28 on page 15.600). If the plot name is empty, the data name in quotation marks can be used, for example:

```
Plot {
   "eTemperatureRelaxationTime"
}
```

Vector data can be plotted by appending `/Vector` to the corresponding keyword, for example:

```
Plot {
   ElectricField/Vector
}
```

Element-based scalar data can be plotted by appending `/Element` to the corresponding keyword, for example:

```
Plot {
   eMobility/Element
}
```

Table 15.167 Scalar data

Data name	Plot name	Location	Description	Unit
AccepMinusConcentration		vertex	Section 2.14	cm^{-3}
AcceptorConcentration	AcceptorConcentration	vertex	Section 2.14	cm^{-3}
AlphaChargeDensity	AlphaCharge	vertex	Section 14.1	cm^{-3}
AlphaGeneration		vertex	G^{Alpha}, (Eq. 15.315)	$cm^{-3}s^{-1}$
AntimonyActiveConcentration		vertex	Section 2.14	cm^{-3}
AntimonyConcentration	AntimonyConcentration	vertex	Sb, Section 2.14	cm^{-3}
AntimonyPlusConcentration	sbPlus	vertex	Sb^+, Chapter 6	cm^{-3}
ArsenicActiveConcentration		vertex	Section 2.14	cm^{-3}
ArsenicConcentration	ArsenicConcentration	vertex	As, Section 2.14	cm^{-3}
ArsenicPlusConcentration	AsPlus	vertex	As^+, Chapter 6	cm^{-3}
AugerRecombination	AugerRecombination	vertex	R^A, (Eq. 15.231)	$cm^{-3}s^{-1}$
AvalancheGeneration	AvalancheGeneration	vertex	$G^{\parallel}$, (Eq. 15.238)	$cm^{-3}s^{-1}$
Band2BandGeneration	Band2Band	vertex	R^{bb}, Section 9.11	$cm^{-3}s^{-1}$
BandGap	BandGap	vertex	E_g, Section 5.2.2	eV
BandgapNarrowing	BandGapNarrowing	vertex	ΔE_g, Section 5.2.2	eV
BeamGeneration	OptBeam	vertex	G^{opt}, (Eq. 15.284)	$cm^{-3}s^{-1}$
BoronActiveConcentration		vertex	Section 2.14	cm^{-3}

Table 15.167 Scalar data

Data name	Plot name	Location	Description	Unit
BoronConcentration	BoronConcentration	vertex	B, Section 2.14	cm^{-3}
BoronMinusConcentration	bMinus	vertex	B^-, Chapter 6	cm^{-3}
BuiltinPotential		vertex	ψ, (Eq. 15.79), (Eq. 15.82), (Eq. 15.83)	V
CDL1Recombination	CDL1	vertex	R_1, Section 9.5	$cm^{-3}s^{-1}$
CDL2Recombination	CDL2	vertex	R_2, Section 9.5	$cm^{-3}s^{-1}$
CDLcRecombination	CDL3	vertex	$R - R_1 - R_2$, Section 9.5	$cm^{-3}s^{-1}$
CDLRecombination	CDL	vertex	R, Section 9.5	$cm^{-3}s^{-1}$
ConductionBandEnergy	ConductionBandEnergy	vertex	E_C, (Eq. 15.62)	eV
ConductionCurrentDensity	ConductionCurrent	vertex	$\lvert\vec{J}_n + \vec{J}_p\rvert$, (Eq. 15.20) or $\lvert\vec{j}_M\rvert$ in metals, (Eq. 15.54)	Acm^{-2}
DeepLevels	DeepLevels	vertex	Section 10.1	cm^{-3}
DielectricConstant		element	ε, (Eq. 15.19)	1
DielectricConstant		vertex	ε, (Eq. 15.19)	1
DielectricConstantAniso		element	ε_{aniso}, Section 20.3	1
DielectricConstantAniso		vertex	ε_{aniso}, Section 20.3	1
DisplacementCurrentDensity	DisplacementCurrent	vertex	$\lvert\vec{J}_D\rvert$	Acm^{-2}
DonorConcentration	DonorConcentration	vertex	Section 2.14	cm^{-3}
DonorPlusConcentration		vertex	Section 2.14	cm^{-3}
DopingConcentration	Doping	vertex	Section 2.14	cm^{-3}
eAlphaAvalanche		vertex	α_n, (Eq. 15.238)	cm^{-1}
eAmorphousRecombination	eGapStatesRecombination	vertex	Chapter 10	$cm^{-3}s^{-1}$
eAmorphousTrappedCharge	eTrappedCharge	vertex	Chapter 10	cm^{-3}
eAugerRecombination		vertex	R_e^A, (Eq. 15.231)	$cm^{-3}s^{-1}$
eAvalancheGeneration	eAvalanche	vertex	G_n, (Eq. 15.238)	$cm^{-3}s^{-1}$
eCDL1Lifetime	eCDL1lifetime	vertex	τ_{n1}, Section 9.5	s
eCDL2Lifetime	eCDL2lifetime	vertex	τ_{n2}, Section 9.5	s
eCurrentDensity	eCurrent	vertex	$\lvert\vec{J}_n\rvert$, (Eq. 15.20)	Acm^{-2}
eDensity	eDensity	vertex	n, (Eq. 15.19)	cm^{-3}
eDirectTunnelCurrent	eDirectTunneling	vertex	Section 16.3.1	Acm^{-2}
eDriftVelocity	eDriftVelocity	vertex	v_n, (Eq. 15.238)	$cm\ s^{-1}$
eeDiffusionLNS		vertex	Table 15.119	$C^2s^{-1}cm^{-1}$
eeDiffusionLNVSD		vertex	Table 15.119	V^2scm^{-3}

Table 15.167 Scalar data

Data name	Plot name	Location	Description	Unit
eEffectiveField	eEffectiveField	vertex	E_n^{eff}, (Eq. 15.248)	V cm^{-1}
eEffectiveStateDensity		vertex	N_C, Section 5.3	cm^{-3}
eeFlickerGRLNS		vertex	Table 15.119	$\text{C}^2\text{s}^{-1}\text{cm}^{-1}$
eeFlickerGRLNVSD		vertex	Table 15.119	$\text{V}^2\text{scm}^{-3}$
eeLNVSD		vertex	Table 15.119	$\text{V}^2\text{scm}^{-3}$
eeMonopolarGRLNS		vertex	Table 15.119	$\text{C}^2\text{s}^{-1}\text{cm}^{-1}$
eeMonopolarGRLNVSD		vertex	Table 15.119	$\text{V}^2\text{scm}^{-3}$
eEnormal	eEnormal	vertex	$F_\perp$, (Eq. 15.165) or $F_{e,\perp}$, (Eq. 15.166)	V cm^{-1}
eEparallel	eEparallel	vertex	F_e, (Eq. 15.190)	V cm^{-1}
eEquilibriumDensity	eEquilibriumDensity	vertex	n, (Eq. 15.19) at zero applied voltages (zero currents)	cm^{-3}
EffectiveBandGap	EffectiveBandGap	vertex	$E_g + \Delta E_g$, Section 5.2.2	eV
EffectiveIntrinsicDensity	EffectiveIntrinsicDensity	vertex	$n_{\text{i, eff}}$, (Eq. 15.103)	cm^{-3}
eGradQuasiFermi	eGradQuasiFermi	vertex	$\lvert\nabla\varphi_e\rvert$, (Eq. 15.191)	V cm^{-1}
eHeatFlux	eHeatFlux	vertex	$\lvert\vec{S}_n\rvert$, (Eq. 15.31)	W cm^{-2}
eInterfaceTrappedCharge		vertex	Chapter 10	cm^{-2}
eIonIntegral	eIonIntegral	vertex	Section 9.10	1
eJouleHeat	eJouleHeat	vertex	Table 15.53	W cm^{-3}
ElectricField	ElectricField	vertex	$\lvert\vec{E}\rvert$	V cm^{-1}
ElectronAffinity	ElectronAffinity	vertex	χ, Section 5.2.2	eV
ElectrostaticPotential	Potential	vertex	ψ, (Eq. 15.19)	V
eLifetime	eLifeTime	vertex	τ_n, (Eq. 15.200)	s
EMLABGeneration		vertex	G^{EMLAB}, Section 13.5	$\text{cm}^{-3}\text{s}^{-1}$
eMobility	eMobility	element	μ_n, Chapter 8	$\text{cm}^2\text{V}^{-1}\text{s}^{-1}$
eMobility	eMobility	vertex	μ_n, Chapter 8	$\text{cm}^2\text{V}^{-1}\text{s}^{-1}$
eMobilityAniso		element	μ_n^{aniso}, Section 20.1	$\text{cm}^2\text{V}^{-1}\text{s}^{-1}$
eMobilityAniso		vertex	μ_n^{aniso}, Section 20.1	$\text{cm}^2\text{V}^{-1}\text{s}^{-1}$
eMobilityAnisoFactor		vertex	r_e, (Eq. 15.407)	1
eMobilityStressFactorXX	eMobilityStressFactorXX	vertex	Section 22.4.1	1
eMobilityStressFactorXY	eMobilityStressFactorXY	vertex	Section 22.4.1	1
eMobilityStressFactorXZ	eMobilityStressFactorXZ	vertex	Section 22.4.1	1
eMobilityStressFactorYY	eMobilityStressFactorYY	vertex	Section 22.4.1	1
eMobilityStressFactorYZ	eMobilityStressFactorYZ	vertex	Section 22.4.1	1

Table 15.167 Scalar data

Data name	Plot name	Location	Description	Unit
eMobilityStressFactorZZ	eMobilityStressFactorZZ	vertex	Section 22.4.1	1
eNLLTunnelingGeneration	eBarrierTunneling	vertex	Section 16.4	$cm^{-3}s^{-1}$
eQuantumPotential	eQuantumPotential	vertex	Λ_n, (Eq. 15.133)	eV
eQuasiFermiPotential	eQuasiFermi	vertex	Φ_n, Section 4.3	V
EquilibriumPotential	EquilibriumPotential	vertex	Ψ, (Eq. 15.19) at zero applied voltages (zero currents)	V
eRelativeEffectiveMass		vertex	m_e, Section 5.3	1
eSaturationVelocity		vertex	$v_{\mathrm{sat,e}}$, Section 8.8.4	$cm\ s^{-1}$
eSaturationVelocityAniso		vertex	$v_{\mathrm{sat,e}}^{\mathrm{aniso}}$, Section 20.1	$cm\ s^{-1}$
eTemperature	eTemperature	vertex	T_n, Section 4.2.4	K
eTemperatureRelaxationTime		vertex	τ_{en}, (Eq. 15.43)	s
eThermoElectricPower	eThermelectricPower	vertex	P_n, (Eq. 15.468)	$V\ K^{-1}$
eVelocity	eVelocity	vertex	$v_n = \left\lvert\frac{\vec{J}_n}{nq}\right\rvert$, (Eq. 15.238)	$cm\ s^{-1}$
FowlerNordheim	FowlerNordheim	vertex	j_{FN}, (Eq. 15.339)	Acm^{-2}
Grad2PoECACGreenFunction		vertex	Table 15.119	$V^2s^2C^{-2}cm^{-2}$
Grad2PoHCACGreenFunction		vertex	Table 15.119	$V^2s^2C^{-2}cm^{-2}$
hAlphaAvalanche		vertex	α_p, (Eq. 15.238)	cm^{-1}
hAmorphousRecombination	hGapStatesRecombination	vertex	Chapter 10	$cm^{-3}s^{-1}$
hAmorphousTrappedCharge	hTrappedCharge	vertex	Chapter 10	cm^{-3}
hAugerRecombination		vertex	R_h^A, (Eq. 15.231)	$cm^{-3}s^{-1}$
hAvalancheGeneration	hAvalanche	vertex	G_p, (Eq. 15.238)	$cm^{-3}s^{-1}$
hCDL1Lifetime	hCDL1lifetime	vertex	τ_{p1}, Section 9.5	s
hCDL2Lifetime	hCDL2lifetime	vertex	τ_{p2}, Section 9.5	s
hCurrentDensity	hCurrent	vertex	$\lvert\vec{J}_p\rvert$, (Eq. 15.20)	Acm^{-2}
hDensity	hDensity	vertex	p, (Eq. 15.19)	cm^{-3}
hDirectTunnelCurrent	hDirectTunneling	vertex	Section 16.3.1	Acm^{-2}
hDriftVelocity	hDriftVelocity	vertex	v_p, (Eq. 15.238)	$cm\ s^{-1}$
HeavyIonChargeDensity	HeavyIonChargeDensity	vertex	Section 14.2	cm^{-3}
HeavyIonGeneration		vertex	G^{HeavyIon}, (Eq. 15.320)	$cm^{-3}s^{-1}$
hEffectiveField	hEffectiveField	vertex	E_p^{eff}, (Eq. 15.249)	Vcm^{-1}
hEffectiveStateDensity		vertex	N_V, Section 5.3	cm^{-3}

Table 15.167 Scalar data

Data name	Plot name	Location	Description	Unit
heiTemperature	HEItemperature	vertex	T_{hei}, hot-electron temperature, computed as postprocessing approach (CarrierTempPost), Chapter 17	K
hEnormal	hEnormal	vertex	$F_{\perp}$, (Eq. 15.165) or $F_{h,\perp}$, (Eq. 15.166)	$\mathrm{V cm^{-1}}$
hEparallel	hEparallel	vertex	F_h, (Eq. 15.190)	$\mathrm{V cm^{-1}}$
hEquilibriumDensity	hEquilibriumDensity	vertex	p, (Eq. 15.19) at zero applied voltages (zero currents)	$\mathrm{cm^{-3}}$
hGradQuasiFermi	hGradQuasiFermi	vertex	$\lvert\nabla\varphi_h\rvert$, (Eq. 15.191)	$\mathrm{V cm^{-1}}$
hhDiffusionLNS		vertex	Table 15.119	$\mathrm{C^2 s^{-1} cm^{-1}}$
hhDiffusionLNVSD		vertex	Table 15.119	$\mathrm{V^2 s cm^{-3}}$
hHeatFlux	hHeatFlux	vertex	$\lvert\vec{S}_p\rvert$, (Eq. 15.32)	$\mathrm{W cm^{-2}}$
hhFlickerGRLNS		vertex	Table 15.119	$\mathrm{C^2 s^{-1} cm^{-1}}$
hhFlickerGRLNVSD		vertex	Table 15.119	$\mathrm{V^2 s cm^{-3}}$
hhLNVSD		vertex	Table 15.119	$\mathrm{V^2 s cm^{-3}}$
hhMonopolarGRLNS		vertex	Table 15.119	$\mathrm{C^2 s^{-1} cm^{-1}}$
hhMonopolarGRLNVSD		vertex	Table 15.119	$\mathrm{V^2 s cm^{-3}}$
hInterfaceTrappedCharge		vertex	Chapter 10	$\mathrm{cm^{-2}}$
hIonIntegral	hIonIntegral	vertex	Section 9.10	1
hJouleHeat	hJouleHeat	vertex	Table 15.53	$\mathrm{W cm^{-3}}$
hLifetime	hLifeTime	vertex	τ_p, (Eq. 15.200)	s
hMobility	hMobility	element	μ_p, Chapter 8	$\mathrm{cm^2 V^{-1} s^{-1}}$
hMobility	hMobility	vertex	μ_p, Chapter 8	$\mathrm{cm^2 V^{-1} s^{-1}}$
hMobilityAniso		element	μ_p^{aniso}, Section 20.1	$\mathrm{cm^2 V^{-1} s^{-1}}$
hMobilityAniso		vertex	μ_p^{aniso}, Section 20.1	$\mathrm{cm^2 V^{-1} s^{-1}}$
hMobilityAnisoFactor		vertex	r_h, (Eq. 15.407)	1
hMobilityStressFactorXX	hMobilityStressFactorXX	vertex	Section 22.4.1	1
hMobilityStressFactorXY	hMobilityStressFactorXY	vertex	Section 22.4.1	1
hMobilityStressFactorXZ	hMobilityStressFactorXZ	vertex	Section 22.4.1	1
hMobilityStressFactorYY	hMobilityStressFactorYY	vertex	Section 22.4.1	1
hMobilityStressFactorYZ	hMobilityStressFactorYZ	vertex	Section 22.4.1	1
hMobilityStressFactorZZ	hMobilityStressFactorZZ	vertex	Section 22.4.1	1
hNLLTunnelingGeneration	hBarrierTunneling	vertex	Section 16.4	$\mathrm{cm^{-3} s^{-1}}$

Table 15.167 Scalar data

Data name	Plot name	Location	Description	Unit
HotElectronInj	HotElectronInjection	vertex	Hot-electron current density j_{he} at interface, (Eq. 15.373), (Eq. 15.379)	Acm^{-2}
HotHoleInj	HotHoleInjection	vertex	Hot-hole current density j_{hh} at interface, (Eq. 15.373), (Eq. 15.379)	Acm^{-2}
hQuantumPotential	hQuantumPotential	vertex	Λ_p, (Eq. 15.133)	eV
hQuasiFermiPotential	hQuasiFermi	vertex	Φ_p, Section 4.3	V
hRelativeEffectiveMass		vertex	m_h, Section 5.3	1
hSaturationVelocity		vertex	$v_{\mathrm{sat,h}}$, Section 8.8.4	$\mathrm{cm\ s}^{-1}$
hSaturationVelocityAniso		vertex	$v_{\mathrm{sat,h}}^{\mathrm{aniso}}$, Section 20.1	$\mathrm{cm\ s}^{-1}$
hTemperature	hTemperature	vertex	T_p, Section 4.2.4	K
hTemperatureRelaxationTime		vertex	τ_{ep}, (Eq. 15.44)	s
hThermoElectricPower	hThermelectricPower	vertex	P_p, (Eq. 15.469)	$\mathrm{V\ K}^{-1}$
hVelocity	hVelocity	vertex	$v_p = \left\lvert \frac{\vec{J}_p}{pq} \right\rvert$, (Eq. 15.238)	$\mathrm{cm\ s}^{-1}$
ImeeDiffusionLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
ImeeFlickerGRLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
ImeeLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
ImeeMonopolarGRLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
ImhhDiffusionLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
ImhhFlickerGRLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
ImhhLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
ImhhMonopolarGRLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
ImLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
IndiumActiveConcentration		vertex	Section 2.14	cm^{-3}
IndiumConcentration	IndiumConcentration	vertex	In, Section 2.14	cm^{-3}
IndiumMinusConcentration	inMinus	vertex	In^-, Chapter 6	cm^{-3}
IntrinsicDensity	IntrinsicDensity	vertex	n_i, (Eq. 15.102)	cm^{-3}
LatticeHeatCapacity		vertex	c, Section 24.1	$\mathrm{JK}^{-1}\mathrm{cm}^{-3}$
LatticeTemperature	LatticeTemperature, Temperature	vertex	T, Section 4.2.3	K
lHeatFlux	lHeatFlux	vertex	$\lvert \vec{S}_L \rvert$, (Eq. 15.33)	Wcm^{-2}
LNVSD		vertex	Table 15.119	$\mathrm{V}^2\mathrm{scm}^{-3}$
MeanIonIntegral	MeanIonIntegral	vertex	Section 9.10	1

Table 15.167 Scalar data

Data name	Plot name	Location	Description	Unit
NDopantActiveConcentration		vertex	Section 2.14	cm^{-3}
NDopantConcentration	NdopantConcentration	vertex	NDopant, Section 2.14	cm^{-3}
NDopantPlusConcentration	NdopantPlus	vertex	$NDopant^+$, Chapter 6	cm^{-3}
NitrogenActiveConcentration		vertex	Section 2.14	cm^{-3}
NitrogenConcentration	NitrogenConcentration	vertex	N, Section 2.14	cm^{-3}
NitrogenPlusConcentration	NitrogenPlus	vertex	N^+, Chapter 6	cm^{-3}
OneOverDegradationTime		vertex	Chapter 11	s^{-1}
OpticalGeneration	OpticalGeneration	vertex	G_0^{opt}, (Eq. 15.304)	$cm^{-3}s^{-1}$
OpticalGenerationFile	OpticalGenerationFile	vertex	Section 13.4	$cm^{-3}s^{-1}$
PDopantActiveConcentration		vertex	Section 2.14	cm^{-3}
PDopantConcentration	pDopantConcentration	vertex	PDopant, Section 2.14	cm^{-3}
PDopantMinusConcentration	pDopantMinus	vertex	$PDopant^-$, Chapter 6	cm^{-3}
PE_Charge	PE_Charge	vertex	q_{PE}, (Eq. 15.675)	cm^{-3}
PeltierHeat	PeltierHeat	vertex	Table 15.53	$W cm^{-3}$
PhosphorusActiveConcentration		vertex	Section 2.14	cm^{-3}
PhosphorusConcentration	PhosphorusConcentration	vertex	P, Section 2.14	cm^{-3}
PhosphorusPlusConcentration	phPlus	vertex	P^+, Chapter 6	cm^{-3}
PiezoFactorN11	PiezoFactorn11	vertex	Π_{11}^n, Section 22.4	$cm^2 dyn^{-1}$
PiezoFactorN12	PiezoFactorn12	vertex	Π_{12}^n, Section 22.4	$cm^2 dyn^{-1}$
PiezoFactorN44	PiezoFactorn44	vertex	Π_{44}^n, Section 22.4	$cm^2 dyn^{-1}$
PiezoFactorP11	PiezoFactorp11	vertex	Π_{11}^p, Section 22.4	$cm^2 dyn^{-1}$
PiezoFactorP12	PiezoFactorp12	vertex	Π_{12}^p, Section 22.4	$cm^2 dyn^{-1}$
PiezoFactorP44	PiezoFactorp44	vertex	Π_{44}^p, Section 22.4	$cm^2 dyn^{-1}$
PMIRecombination	PMIRecombination	vertex	R^{PMI}, Section 33.7	$cm^{-3}s^{-1}$
PMIUserField0	PMIUserField0	vertex	Section 33.4	1
PMIUserField1	PMIUserField1	vertex	Section 33.4	1
PMIUserField2	PMIUserField2	vertex	Section 33.4	1
PMIUserField3	PMIUserField3	vertex	Section 33.4	1
PMIUserField4	PMIUserField4	vertex	Section 33.4	1
PMIUserField5	PMIUserField5	vertex	Section 33.4	1
PMIUserField6	PMIUserField6	vertex	Section 33.4	1
PMIUserField7	PMIUserField7	vertex	Section 33.4	1
PMIUserField8	PMIUserField8	vertex	Section 33.4	1

Table 15.167 Scalar data

Data name	Plot name	Location	Description	Unit
PMIUserField9	PMIUserField9	vertex	Section 33.4	1
PoECImACGreenFunction		vertex	Table 15.119	VsC^{-1}
PoECReACGreenFunction		vertex	Table 15.119	VsC^{-1}
PoETImACGreenFunction		vertex	Table 15.119	A^{-1}
PoETReACGreenFunction		vertex	Table 15.119	A^{-1}
PoHCImACGreenFunction		vertex	Table 15.119	VsC^{-1}
PoHCReACGreenFunction		vertex	Table 15.119	VsC^{-1}
PoHTImACGreenFunction		vertex	Table 15.119	A^{-1}
PoHTReACGreenFunction		vertex	Table 15.119	A^{-1}
Polarization	Polarization	vertex	$\lvert\vec{P}\rvert$, Chapter 21	Ccm^{-2}
QuasiFermiPotential		vertex	Φ, (Eq. 15.92)	V
RadiationGeneration		vertex	G_r, (Eq. 15.280)	$\mathrm{cm}^{-3}\mathrm{s}^{-1}$
RadiativeRecombination	RadiativeRecombination	vertex	R, (Eq. 15.230)	$\mathrm{cm}^{-3}\mathrm{s}^{-1}$
RecombinationHeat	RecombinationHeat	vertex	Table 15.53	Wcm^{-3}
ReeeDiffusionLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
ReeeFlickerGRLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
ReeeLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
ReeeMonopolarGRLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
RefractiveIndex		element	n, Section 13.3.5	1
RefractiveIndex		vertex	n, Section 13.3.5	1
RehhDiffusionLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
RehhFlickerGRLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
RehhLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
RehhMonopolarGRLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
ReLNVXVSD		vertex	Table 15.120	$\mathrm{V}^2\mathrm{scm}^{-3}$
SpaceCharge	SpaceCharge	vertex	(Eq. 15.19)	cm^{-3}
SRHRecombination	SRHRecombination	vertex	R^{SRH}, Section 9.1	$\mathrm{cm}^{-3}\mathrm{s}^{-1}$
StressXX	Stressxx	vertex	Chapter 22	Pa
StressXY	Stressxy	vertex	Chapter 22	Pa
StressXZ	Stressxz	vertex	Chapter 22	Pa
StressYY	Stressyy	vertex	Chapter 22	Pa
StressYZ	Stressyz	vertex	Chapter 22	Pa
StressZZ	Stresszz	vertex	Chapter 22	Pa

Table 15.167 Scalar data

Data name	Plot name	Location	Description	Unit
SurfaceMultRecombination	SurfaceMultRecombination	vertex	Section 9.4	$\text{cm}^{-3}\text{s}^{-1}$
SurfaceRecombination	SurfaceRecombination	vertex	Section 9.4	$\text{cm}^{-3}\text{s}^{-1}$
ThermalConductivity	ThermalConductivity	vertex	κ, (Eq. 15.467)	$\text{Wcm}^{-1}\text{K}^{-1}$
ThermalConductivityAniso		vertex	κ_{aniso}, Section 20.4	$\text{Wcm}^{-1}\text{K}^{-1}$
ThomsonHeat	ThomsonHeat	vertex	Table 15.53	Wcm^{-3}
TotalConcentration		vertex	Section 2.14	cm^{-3}
TotalCurrentDensity	Current	vertex	$\left\vert\vec{J}_n+\vec{J}_p+\vec{J}_D\right\vert$	Acm^{-2}
TotalHeat	TotalHeat	vertex	Sum of all heat generation terms, Section 4.2.3, Section 4.2.4, Section 4.2.5	Wcm^{-3}
TotalInterfaceTrapConcentration		vertex	Chapter 10	cm^{-2}
TotalRecombination	TotalRecombination	vertex	Sum of all generation–recombination terms, Chapter 9	$\text{cm}^{-3}\text{s}^{-1}$
TotalTrapConcentration		vertex	Chapter 10	cm^{-3}
ValenceBandEnergy	ValenceBandEnergy	vertex	E_V, (Eq. 15.63)	eV
xMoleFraction	xMoleFraction	vertex	Section 18.4	1
yMoleFraction	yMoleFraction	vertex	Section 18.4	1

Table 15.168 Vector data

Data name	Plot name	Location	Description	Unit
ConductionCurrentDensity	ConductionCurrent	vertex	$\vec{J}_n+\vec{J}_p$, (Eq. 15.20) or $\vec{j}_M$ in metals, (Eq. 15.54)	Acm^{-2}
DisplacementCurrentDensity	DisplacementCurrent	vertex	$\vec{J}_D$	Acm^{-2}
eCurrentDensity	eCurrent	vertex	$\vec{J}_n$, (Eq. 15.20)	Acm^{-2}
eDriftVelocity	eDriftVelocity	vertex	$\vec{v}_n$, (Eq. 15.238)	cm s^{-1}
eGradQuasiFermi	eGradQuasiFermi	vertex	$\nabla\varphi_e$, (Eq. 15.191)	Vcm^{-1}
eHeatFlux	eHeatFlux	vertex	$\vec{S}_n$, (Eq. 15.31)	Wcm^{-2}
ElectricField	ElectricField	vertex	$\vec{E}$	Vcm^{-1}
EquilibriumElectricField		vertex	E_{eq}, (Eq. 15.86)	Vcm^{-1}
eVelocity	eVelocity	vertex	$\vec{v}_n = \frac{\vec{J}_n}{nq}$, (Eq. 15.238)	cm s^{-1}
GradPoECImACGreenFunction		vertex	Table 15.119	$\text{VsC}^{-1}\text{cm}^{-1}$
GradPoECReACGreenFunction		vertex	Table 15.119	$\text{VsC}^{-1}\text{cm}^{-1}$

Table 15.168 Vector data

Data name	Plot name	Location	Description	Unit
GradPoETImACGreenFunction		vertex	Table 15.119	$A^{-1}cm^{-1}$
GradPoETReACGreenFunction		vertex	Table 15.119	$A^{-1}cm^{-1}$
GradPoHCImACGreenFunction		vertex	Table 15.119	$VsC^{-1}cm^{-1}$
GradPoHCReACGreenFunction		vertex	Table 15.119	$VsC^{-1}cm^{-1}$
GradPoHTImACGreenFunction		vertex	Table 15.119	$A^{-1}cm^{-1}$
GradPoHTReACGreenFunction		vertex	Table 15.119	$A^{-1}cm^{-1}$
hCurrentDensity	hCurrent	vertex	$\vec{J}_p$, (Eq. 15.20)	Acm^{-2}
hDriftVelocity	hDriftVelocity	vertex	$\vec{v}_p$, (Eq. 15.238)	$cm\ s^{-1}$
hGradQuasiFermi	hGradQuasiFermi	vertex	$\nabla\varphi_h$, (Eq. 15.191)	Vcm^{-1}
hHeatFlux	hHeatFlux	vertex	$\vec{S}_p$, (Eq. 15.32)	Wcm^{-2}
hVelocity	hVelocity	vertex	$\vec{v}_p = \frac{\vec{J}_p}{pq}$, (Eq. 15.238)	$cm\ s^{-1}$
lHeatFlux	lHeatFlux	vertex	$\vec{S}_L$, (Eq. 15.33)	Wcm^{-2}
NonLocalBackDirection	NonLocal	vertex	Section 2.10.7.2	μm
NonLocalDirection	NonLocal	vertex	Section 2.10.7.2	μm
PE_Polarization	PE_Polarization	vertex	P_{PE}, (Eq. 15.675)	Ccm^{-2}
Polarization	Polarization	element	$\vec{P}$, Chapter 21	Ccm^{-2}
Polarization	Polarization	vertex	$\vec{P}$, Chapter 21	Ccm^{-2}
TotalCurrentDensity	Current	vertex	$\vec{J}_n + \vec{J}_p + \vec{J}_D$	Acm^{-2}

Bibliography

[1] R. E. Bank, D. J. Rose, and W. Fichtner, "Numerical Methods for Semiconductor Device Simulation," *IEEE Transactions on Electron Devices*, vol. ED-30, pp. 1031–1041, 1983.

[2] G. Wachutka, "An extended thermodynamic model for the simultaneous simulation of the thermal and electrical behavior of semiconductor devices," in *Proceedings of the Sixth International NASECODE Conference* (J. J. H. Miller, ed.), Boole Press Ltd., pp. 409–414, 1989.

[3] A. Benvenuti, G. Ghione, M. R. Pinto, J. W. M. Coughran, and N. L. Schryer, "Coupled thermal-fully hydrodynamic simulation of InP-based HBTs," in *IEDM Technical Digest*, pp. 737–740, 1992.

[4] A. Schenk and S. Müller, "Analytical Model of the Metal-Semiconductor Contact for Device Simulation," in *Simulation of Semiconductor Devices and Processes*, vol. 5, pp. 441–444, Sept. 7–9, Vienna, Austria, 1993.

[5] A. Liegmann, "The application of supernodal techniques on the solution of structurally symmetric systems," Technical Report 92/5, Integrated Systems Laboratory, ETH Zurich, Switzerland, 1992.

[6] T.-W. Tang, "Extension of the Scharfetter-Gummel algorithm to the energy balance equation," *IEEE Transactions on Electron Devices*, vol. ED-31, no. 12, pp. 1912–1914, 1984.

[7] C. C. McAndrew, K. Singhal, and E. L. Heasell, "A consistent nonisothermal extension of the Scharfetter-Gummel stable difference approximation," *IEEE Electron Device Letters*, vol. EDL-6, no. 9, pp. 446–447, 1985.

[8] B. Meinerzhagen, K. B. Bach, I. Bork, and W. Engl, "A new highly efficient nonlinear relaxation scheme for hydrodynamic MOS simulations," *NUPAD IV Conf. Dig. Tech. Papers*, pp. 91–96, 1992.

[9] Y. Apanovich, E. Lyumkis, B. Polsky, and P. Blakey, "An investigation of coupled and decoupled iterative algorithms for energy balance calculations," in *Proc. SISDEP V*, Zurich, Switzerland, pp. 233–236, 1993.

[10] Y. Apanovich, P.Blakey, R.Cottle, E. Lyumkis, B. Polsky, A.Shur,and A.Tcherniaev, "Numerical simulation of submicrometer devices including coupled nonlocal transport and nonisothermal effects," *IEEE Transactions on Electron Devices*, vol. ED-42, pp. 890–898, May, 1995.

[11] H. B. Callen, *Thermodynamics and an Introduction to Thermostatistics*. New York: John Wiley & Sons, 1985.

[12] R. Stratton, "Diffusion of hot and cold electrons in semiconductor barriers," *Phys. Rev.*, vol. 126, no. 6, pp. 2002–2014, 1962.

[13] K. Bløtekjær, "Transport equations for electrons in two-valley semiconductors," *IEEE Transactions on Electron Devices*, vol. ED-17, no. 1, pp. 38–47, 1970.

[14] A. Benvenuti, G. Ghione, and C. U. Naldi, "Non-stationary transport hbt modeling under non-isothermal conditions," in *SISDEP-5*, Vienna, pp. 453–456, Sept., 1993.

[15] J. W. Roberts and S. G. Chamberlain, "Energy-momentum transport model suitable for small geometry silicon device simulation," *COMPEL*, vol. 9, no. 1, pp. 1–22, 1990.

[16] A. Benvenuti, M. R. Pinto, J. W. M. Coughran, N. L. Schryer, C. U. Naldi, and G. Ghione, "Evaluation of the influence of convective energy in hbts using a fully-hydrodynamic model," in *IEDM Technical Digest*, pp. 499–502, 1991.

[17] A. Benvenuti, F. Bonani, G. Ghione, C. U. Naldi, M. Kärner, and K. Schaper, “Analysis of output ndr in power AlGaAs/GaAs hbts by means of a thermal-fully hydrodynamic model,” in *ISDRS 93 Proceedings*, pp. 499–502, 1993.

[18] S. Szeto and R. Reif, “A unified electrothermal hot-carrier transport model for silicon bipolar transistor simulation,” *Solid-State Electronics*, vol. 32, no. 4, pp. 307–315, 1989.

[19] A. Pierantoni, A. Liuzzo, P. Ciampolini, and G. Baccarani, “Three-dimensional implementation of a unified transport model,” in *SISDEP*, pp. 125–128, 1993.

[20] D. Chen, Z. Yu, K.-C. Wu, R. Goosens, and R. W. Dutton, “Dual energy transport model with coupled lattice and carrier temperatures,” in *SISDEP-5*, Vienna, pp. 157–160, Sept., 1993.

[21] “Six-month technical report, period 1 (May 27 – November 26, 1992),” in *ESPRIT-6075 (DESSIS) Project Report*, 1992.

[22] A. Bringer and G. Schön, “Extended moment equations for electron transport in semiconducting submicron structures,” *Journal of Applied Physics*, vol. 64, no. 5, pp. 2447–55, 1988.

[23] M. A. Stettler, M. A. Alam, and M. S. Lundstrom, “A critical examination of the assumptions underlying macroscopic transport equations for silicon devices,” *IEEE Transactions on Electron Devices*, vol. 40, no. 4, pp. 733–740, 1983.

[24] B. Baccarani and M. R. Wordeman, “An investigation of steady-state velocity overshoot in silicon,” *Solid-State Electronics*, vol. 28, no. 4, pp. 407–416, 1985.

[25] E. M. Azoff, “Semiclassical high-field transport equations for nonparabolic heterostructure degenerate semiconductors,” *Journal of Applied Physics*, vol. 64, no. 5, pp. 2439–2446, 1988.

[26] M. Stecher, B. Meinerzhagen, I. Bork, and W. L. Engl, “On the influence of thermal diffusion and heat flux on bipolar device and circuit performance,” in *SISDEP-5*, Vienna, pp. 49–52, Sept., 1993.

[27] M. C. Vecchi and L. G. Reyna, “Generalized energy transport models for semiconductor device simulation,” *Solid-State Electronics*, vol. 37, no. 10, pp. 1705–1716, 1994.

[28] Y. Apanovich, E. Lyumkis, B. Polsky, A. Shur, and P. Blakey, “Steady-state and transient analysis of submicron devices using energy balance and simplified hydrodynamic models,” *IEEE Transactions on CAD*, vol. 13, pp. 702–710, June, 1994.

[29] D. Chen, E. Sangiori, M. R. Pinto, E. C. Kan, U. Ravaioli, and R. W. Dutton, “An improved energy transport model including nonparabolicity and non-Maxwellian distribution effects,” *IEEE Transactions on Electron Devices*, vol. ED-39, pp. 26–28, January, 1992.

[30] G. Wachutka, “Rigorous thermodynamic treatment of heat generation and conduction in semiconductor device modeling,” *IEEE Trans.*, vol. CAD-9, pp. 1141–1149, 1990.

[31] J. A. Andrews, “Package thermal resistance model: Dependency on equipment design,” *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, vol. 11, pp. 528–537, December, 1988.

[32] J. N. Sweet and W. T. Cooley, “Thermal resistance measurement and finite element calculations for ceramic hermetic packages,” in *Proc. Sixth IEEE SEMI-THERM Symposium*, pp. 10–16, 1990.

[33] T. Hopkins, C. Cognetti, and R. Tiziani, “Designing with thermal impedance,” in *Proc. Fourth IEEE SEMI-THERM Symposium*, pp. 55–61, 1988.

[34] R. L. Kozarek, “Effect of case temperature measurement errors on the junction-to-case thermal resistance of a ceramic PGA,” in *Proc. Sixth IEEE SEMI-THERM Symposium*, pp. 44–51, 1991.

[35] Y. J. Min, A. L. Palisoc, and C. C. Lee, “Transient thermal study of semiconductor devices,” *IEEE Trans. Components, Hybrids, and Manufacturing Technology*, vol. 13, no. 4, pp. 980–988, 1990.

[36] F. Curatelli and G. M. Bisio, “Characterization of the thermal behaviour in ICs,” *Solid-State Electronics*, vol. 34, no. 7, pp. 751–760, 1991.

[37] G. N. Ellison, “Tams: A thermal analyzer for multilayer structures,” *Electrosoft*, vol. 1, no. 2, pp. 85–97, 1990.

[38] R. A. Tatara, “Thermal modeling previews electronic device performance,” *PCIM*, pp. 9–21, October 1991.

[39] S. Song and M. M. Yovanovich, “Relative contact pressure: Dependence on surface roughness and vickers microhardness,” *AIAA Journal of Thermophysics and Heat Transfer*, vol. 2, pp. 43–47, January, 1988.

[40] V. P. Deshwal, B. B. Dixit, K. M. K. Srivatsa, P. D. Vyas, and W. S. Khokle, “Optimum thickness determination of the electrode for large silicon power devices and its improved bonding with silicon wafer having n-doped substrate,” *Indian Journal of Technology*, vol. 29, pp. 395–398, August, 1991.

[41] W. S. Childres and G. P. Peterson, “Quantification of thermal contact conductance in electronic packages,” in *Proc. Fifth IEEE SEMI-THERM Symposium*, pp. 30–36, 1989.

[42] D. J. Dean, *Thermal Design of Electronic Circuit Boards and Packages*, Ayr, Scotland: Electrochemical Publications, Ltd., 1985.

[43] G. N. Ellison, “Theoretical calculation of the thermal resistance of a conducting and convecting surface,” *IEEE Trans. Parts, Hybrids, and Packaging*, vol. PHP-12, pp. 265–266, September, 1976.

[44] S. N. Rea and S. E. West, “Thermal radiation from finned heat sinks,” *IEEE Trans. Parts, Hybrids, and Packaging*, vol. PHP-12, pp. 115–117, September 1976.

[45] L. Buller and B. McNelis, “Effects of radiation on enhanced electronic cooling,” *IEEE Trans. Components, Hybrids, and Manufacturing Technology*, vol. 11, pp. 538–544, December, 1988.

[46] C. Zardini, F. Rodes, G. Duchamp, and J.-L. Aucouturier, “3d thermal simulation of power hybrid assemblies,” *Hybrid Circuits*, pp. 20–22, January, 1991.

[47] I. Hirsch, E. Berman, and N. Haik, “Thermal resistance evaluation in 3D thermal simulation of MOSFET transistors,” *Solid-State Electronics*, pp. 106–108, January, 1991.

[48] M. A. Green, “Intrinsic concentration, effective densities of states, and effective mass in Silicon,” *Journal of Applied Physics*, vol. 67, no. 6, pp. 2944–54, 1990.

[49] J. E. Lang, F. L. Madarasz, and P. M. Hemeger, “Temperature dependent density of states effective mass in nonparabolic p-type Silicon,” *Journal of Applied Physics*, vol. 54, no. 6, p. 3612, 1983.

[50] W. Bludau, A. Onton, and W. Heinke, “Temperature dependence of the band gap in Silicon,” *Journal of Applied Physics*, vol. 45, no. 4, pp. 1846–1848, 1974.

[51] J. W. Slotboom and H. C. de Graaff, “Measurements of Bandgap Narrowing in Si Bipolar Transistors,” *Solid-State Electronics*, vol. 19, pp. 857–862, 1976.

[52] J. W. Slotboom and H. C. de Graaff, “Bandgap Narrowing in Silicon Bipolar Transistors,” *IEEE Transactions on Electron Devices*, vol. ED-24, no. 8, pp. 1123–1125, 1977.

[53] J. W. Slotboom, “The pn-Product in Silicon,” *Solid-State Electronics*, vol. 20, pp. 279–283, 1977.

[54] D. B. M. Klaassen, J. W. Slotboom, and H. C. de Graaff, “Unified apparent bandgap narrowing in n- and p-type Silicon,” *Solid-State Electronics*, vol. 35, no. 2, pp. 125–129, 1992.

[55] J. del Alamo, S. Swirhun, and R. M. Swanson, “Simultaneous measuring of hole lifetime, hole mobility and bandgap narrowing in heavily doped n-type Silicon,” in *IEDM Technical Digest*, pp. 290–293, December 1985.

[56] J. del Alamo, S. Swirhun, and R. M. Swanson, “Measuring and modeling minority carrier transport in heavily doped Silicon,” *Solid-State Electronics*, vol. 28, no. 1, pp. 47–54, 1985.

[57] S. E. Swirhun, Y.-H. Kwark, and R. M. Swanson, "Measurement of electron lifetime, electron mobility and bandgap narrowing in heavily doped p-type Silicon," in *IEDM Technical Digest*, pp. 24–27, December 1986.

[58] S. E. Swirhun, J. A. del Alamo, and R. M. Swanson, "Measurement of hole mobility in heavily doped n-type Silicon," *IEEE Electron Device Letters*, vol. EDL-7, no. 3, pp. 168–171, 1986.

[59] J. del Alamo and R. M. Swanson, "Measurement of steady-state minority carrier transport parameters in heavily doped n-type Silicon," *IEEE Transactions on Electron Devices*, vol. ED-34, no. 7, pp. 1580–1589, 1987.

[60] H. S. Bennett and C. L. Wilson, "Statistical Comparisons of Data on Band-Gap Narrowing in Heavily Doped Silicon: Electrical and Optical Measurements," *Journal of Applied Physics*, vol. 55, no. 10, pp. 3582–3587, 1984.

[61] C. Lombardi, S. Manzini, A. Saporito, and M. Vanzi, "A Physically Based Mobility Model for Numerical Simulation of Nonplanar Devices," *IEEE Transactions on CAD*, vol. 7, no. 11, pp. 1164–1171, 1988.

[62] G. Masetti, M. Severi, and S. Solmi, "Modeling of carrier mobility against carrier concentration in Arsenic-, Phosphorus- and Boron-doped Silicon," *IEEE Transactions on Electron Devices*, vol. ED-30, pp. 764–769, 1983.

[63] D. M. Caughey and R. E. Thomas, "Carrier mobilities in Silicon empirically related to doping and field," *Proc. IEEE*, pp. 2192–2193, Dec. 1967.

[64] S. C. Choo, "Theory of a Forward-Biased Diffused-Junction P-L-N Rectifier. Part I: Exact Numerical Solutions," *IEEE Transactions on Electron Devices*, vol. ED-19, no. 8, pp. 954–966, 1972.

[65] N. H. Fletcher, "The high current limit for semiconductor junction devices," *Proceedings of Institute of Radio Engineers*, vol. 45, pp. 862–872, 1957.

[66] D. B. M. Klaassen, "A unified mobility model for device simulation – I. Model equations and concentration dependence," *Solid-State Electronics*, vol. 35, no. 7, pp. 953–959, 1992.

[67] C. Canali, G. Majni, R. Minder, and G. Ottaviani, "Electron and hole drift velocity measurements in Silicon and their empirical relation to electric field and temperature," *IEEE Transactions on Electron Devices*, vol. ED-22, pp. 1045–1047, 1975.

[68] B. Meinerzhagen and W. L. Engl, "The influence of the thermal equilibrium approximation on the accuracy of classical two-dimensional numerical modeling of silicon submicrometer mos transistors," *IEEE Transactions Electron Devices*, vol. 35, no. 5, pp. 689–697, 1988.

[69] D. Kendall. presented at the Conf. Physics and Application of Lithium Diffused Silicon, NASA, Goddard Space Flight Center, December, 1969.

[70] J. G. Fossum, "Computer-aided numerical analysis of Silicon solar cells," *Solid-State Electronics*, vol. 19, pp. 269–277, 1976.

[71] J. G. Fossum and D. S. Lee, "A physical model for the dependence of carrier lifetime on doping density in nondegenerate Silicon," *Solid-State Electronics*, vol. 25, no. 8, pp. 741–747, 1982.

[72] J. G. Fossum, R. P. Mertens, D. S. Lee, and J. F. Nijs, "Carrier recombination and lifetime in highly doped Silicon," *Solid-State Electronics*, vol. 26, no. 6, pp. 569–576, 1983.

[73] A. Schenk, "A model for the field and temperature dependence of Shockley-Read-Hall lifetimes in Silicon," *Solid-State Electronics*, vol. 35, no. 11, pp. 1585–1596, 1992.

[74] M. S. Tyagi and R. van Overstraeten, "Minority carrier recombination in heavily-doped Silicon," *Solid-State Electronics*, vol. 26, no. 6, pp. 577–597, 1983.

[75] H. Goebel and K. Hoffmann, "Full dynamic power diode model including temperature behavior for use in circuit simulators," in *Proceedings of 1992 International Symposium on Power Semiconductor Devices & ICs*, (Tokyo), pp. 130–135, 1992.

[76] U. Lindefelt. ABB Corporate Research, Västerås, Sweden. private communication.

[77] R. R. King, R. A. Sinton, and R. M. Swanson, “Studies of diffused phosphorus emitters: Saturation current, surface recombination velocity, and quantum efficiency,” *IEEE Transactions on Electron Devices*, vol. 37, pp. 365–371, 1990.

[78] R. R. King and R. M. Swanson, “Studies of diffused boron emitters: Saturation current, bandgap narrowing, and surface recombination velocity,” *IEEE Transactions on Electron Devices*, vol. 38, pp. 1399–1409, 1991.

[79] A. Cuevas, P. A. Basore, G. Giroult-Matlakowski, and C. DuBois, “Surface recombination velocity and bandgap narrowing of highly doped n-type silicon,” in *Proceedings of the 13th European Photovoltaic Solar Energy Conference*, (Nice, France), Oct. 1995. To be published.

[80] A. Schenk and U. Krumbein, “Coupled Defect-Level Recombination: Theory and Application to Anomalous Diode Characteristics,” *Journal of Applied Physics*, vol. 77, no. 17, p. n.n., 1995.

[81] L. Huldt, N. G. Nilsson, and K. G. Svantesson, “The temperature dependence of band-to-band Auger recombination in silicon,” *Applied Physics Letters*, vol. 35, no. 10, p. 776, 1979.

[82] W. Lochmann and A. Haug, “Phonon-assisted Auger recombination in Si with direct calculation of the overlap integrals,” *Solid-State Communications*, vol. 35, pp. 553–556, 1980.

[83] R. Häcker and A. Hangleiter, “Intrinsic upper limits of the carrier lifetime in silicon,” *Journal of Applied Physics*, vol. 75, pp. 7570–7572, 1994.

[84] A. G. Chynoweth, “Ionization rates for electrons and holes in Silicon,” *Phys. Rev.*, vol. 109, no. 5, pp. 1537–1540, 1958.

[85] R. V. Overstraeten and H. D. Man, “Measurement of the ionization rates in diffused Silicon p-n junctions,” *Solid-State Electronics*, vol. 13, pp. 583–608, 1970.

[86] Y. Okuto and C. R. Crowell, “Threshold energy effects on avalanche breakdown voltage in semiconductor junctions,” *Solid-State Electronics*, vol. 18, pp. 161–168, 1975.

[87] T. Lackner, “Avalanche Multiplication in Semiconductors: A Modification of Chynoweth’s Law,” *Solid-State Electronics*, vol. 34, pp. 33–42, 1991.

[88] A. Schenk, “Rigorous theory and simplified model of the band-to-band tunneling in Silicon,” *Solid-State Electronics*, vol. 36, no. 1, pp. 19–34, 1993.

[89] A. Erlebach, “Parameter für die Ladungsträgergeneration bei Alphateilcheneinfall in Silizium,” private communication.

[90] L. C. Northcliffe and R. F. Schilling, “Range and stopping-power tables for heavy ions,” *Nuclear Data Tables*, vol. A7, pp. 233–463, 1970.

[91] C. Fiegna, F. Venturi, M. Melanotte, E. Sangiorgi, and B. Riccò, “Simple and efficient modeling of EPROM writing,” *IEEE Transactions on Electron Devices*, vol. ED-38, no. 3, pp. 603–610, 1991.

[92] S. M. Sze, *Physics of Semiconductor Devices*, John Wiley & Sons, 2nd ed., 1981.

[93] C. J. Glassbrenner and G. A. Slack, “Thermal conductivity of silicon and germanium from 3 K to the melting point,” *Physical Review*, vol. 134, pp. A1058–A1069, May, 1964.

[94] S. S. Furkay, “Thermal characterization of plastic and ceramic surface-mount components,” *IEEE Trans. Components, Hybrids, and Manufacturing Technology*, vol. 11, pp. 521–527, December, 1988.

[95] R. A. Smith, *Semiconductors*, Cambridge: University Press, 2nd ed., 1978.

[96] C. Herring, “The role of low-frequency phonons in thermoelectricity and thermal conduction,” in *Proc. Internat. Colloq. “Semiconductors and Phosphors”* at Garmisch Partenkirchen, 1956.

[97] T. H. Geballe and G. W. Hull, “Seebeck effect in silicon,” *Physical Review*, vol. 98, pp. 941–947, 1955.

[98] W. Fulkerson, J. P. Moore, R. K. Williams, R. S. Graves, and D. L. McElroy, “Thermal conductivity, electrical resistivity, and seebeck coefficient of Silicon from 100 to 1300 K,” *Phys. Rev.*, vol. 167, no. 3, pp. 765–782, 1968.

[99] R. S. Varga, *Matrix Iterative Analysis*, Englewood Cliffs: Prentice-Hall, 1962.

[100] E. M. Buturla, P. E. Cottrell, B. M. Grossman, and K. A. Salsburg, “Finite-element analysis of semiconductor devices: The FIELDAY program,” *IBM J. Res. Develop.*, vol. 25, pp. 218–239, 1981.

[101] R. E. Bank, W. M. Coughran, Jr., W. Fichtner, E. H. Grosse, D. J. Rose, and R. K. Smith, “Transient simulation of silicon devices and circuits,” *IEEE Trans.*, vol. CAD-4, pp. 436–451, 1985.

[102] R. E. Bank and D. J. Rose, “Global Approximate Newton Methods,” *Numer. Math.*, vol. 37, pp. 279–295, 1981.

[103] W. Allegretto, A. Nathan, and H. Baltes, “Numerical Analysis of Magnetic-Field-Sensitive Bipolar Devices,” *IEEE Transactions on CAD*, vol. CAD-10, pp. 501–511, 1991.

[104] C. Riccobene, G. Wachutka, J. F. Bürgler, and H. Baltes, “Operating Principle of Dual Collector Magnetotransistors Studied by Two-Dimensional Simulation,” *IEEE Transactions on Electron Devices*, vol. ED-41, pp. 1136–1148, 1994.

[105] C. Riccobene, K. Gärtner, G. Wachutka, H. Baltes, and W. Fichtner, “First Three-Dimensional Numerical Analysis of Magnetic Vector Probe,” in *IEDM Technical Digest*, San Francisco, California, USA, pp. 727–730, 1994.

[106] M. Lades, J. Frank, J. Funk, and G. Wachutka, “Analysis of Piezoresistive Effects in Silicon Structures Using Multidimensional Process and Device Simulation,” in *SISDEP-6*, (Erlangen), pp. 82–85, September, 1995.

[107] Z. Z. Wang, *Modélisation de la piézorésistivité du Silicium*, Ph.D. thesis, University of Science and Technology, Lille, France, 1994.

[108] J. F. Nye, *Physical Properties of Crystals*, Oxford: Clarendon Press, 1985.

[109] Y. Kanda, “A Graphical Representation of the Piezoresistance Coefficients in Silicon,” *IEEE Transactions on Electron Devices,* vol. 29, pp. 64–70, 1982.

[110] S. Selberherr, *Analysis and Simulation of Semiconductor Devices*, Wien: Springer, 1984.

[111] S. Wolfram, *Mathematica*, Redwood City: Addison-Wesley, 1991.

[112] M. J. van Dort, P. H. Woerlee, and A. J. Walker, “A Simple Model for Quantization Effects in Heavily-Doped Silicon MOSFETs at Inversion Conditions,” *Solid-State Electronics*, vol. 37, no. 3, pp. 411–414, 1994.

[113] S. A. Hareland, S. Jallepalli, G. Chindalore, W.-K. Shih, A. F. Tasch, Jr., and C. M. Maziar, “A Simple Model for Quantum Mechanical Effects in Hole Inversion Layers in Silicon PMOS Devices,” *IEEE Transactions on Electron Devices*, vol. 44, no. 7, pp. 1172–1173, 1997.

[114] J. J. Liou, “Modeling the Tunneling Current in Reverse-Biased p/n Junctions”, *Solid-State Electronics*, vol. 33, no. 7, pp. 971–972, 1990.

[115] K. Hasnat, C.-F. Yeap, W.-K. Shih, S. A. Hareland, V. M. Agostinelli, A. F. Tasch, and C. M. Mazair, “A Pseudo-Lucky Electron Model for Simulation of Electron Gate Current in Submicron NMOSFET’s,” *IEEE Transactions on Electron Devices*, vol. 43, no. 8, pp. 1264–1273, 1996.

[116] M. N. Darwish, J. L. Lentz, M. R. Pinto, P. M. Zeitzoff, T. J. Krutsick, and H. H. Vuong, “An Improved Electron and Hole Mobility Model for General Purpose Device Simulation,” *IEEE Transactions on Electron Devices*, vol. 44, no. 9, pp. 1529–1538, 1997.

[117] N. D. Arora, J. R. Hauser, and D. J. Roulston, “Electron and Hole Mobilities in Silicon as a Function of Concentration and Temperature,” *IEEE Transactions on Electron Devices*, vol. ED-29, pp. 292–295, 1982.

[118] A. Schenk and G. Heiser, “Modeling and Simulation of Tunneling through Ultra-Thin Gate Dielectrics,” *Journal of Applied Physics*, vol. 81, no. 12, p.7900.

[119] F. Bonani, G. Ghione, M.R. Pinto, and R.K. Smith, “An Efficient Approach to Noise Analysis through Multidimensional Physics-based Models,” *IEEE Transactions on Electron Devices*, vol.45, pp. 261–269, 1998.

[120] J.-P. Nougier, “Fluctuations and Noise of Hot Carriers in Semiconductor Materials and Devices,” *IEEE Transactions on Electron Devices*, vol.41, pp. 2034–2049, 1994.

[121] B. Jiang, P. Zurcher, R. E. Jones, S. J. Gillespie, and J. C. Lee, “Computationally Efficient Ferroelectric Capacitor Model for Circuit Simulation,” in *Symposium on VLSI Technology Digest of Technical Papers*, pp. 141–142, 1997.

[122] K. Dragosits, *Modeling and Simulation of Ferroelectric Devices*, Ph.D. thesis, Fakultät für Elektrotechnik und Informationstechnik, Technische Universität Wien, 2000.

[123] J.-L. Leray, “Total Dose Effects: Modeling For Present And Future,” *IEEE NSREC Short Course*, 1999.

[124] D. Schroeder, *Modelling of Interface Carrier Transport for Device Simulation*, Springer, 1994.

[125] K. Horio and H. Yanai, “Numerical Modeling of Heterojunctions Including the Thermionic Emission Mechanism at Heterojunction Interface,” *IEEE Transactions on Electron Devices*, vol. ED-37, pp. 1093–1098, 1990.

[126] G. A. M. Hurkx, D. B. M. Klassen, and M. P. G. Knuvers, “A New Recombination Model for Device Simulation Including Tunneling,” *IEEE Transactions on Electron Devices*, vol. ED-39, pp. 331–338, 1992.

[127] S. Sugino, N. Takakura, D. Chen, and R. W. Dutton, “Analysis of Writing and Erasing Procedure of Flotox EEPROM Using the New Charge Balance Condition (CBC) Model,” in *Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits: NUPAD IV,* Seattle, WA, pp. 65–69, 1992.

[128] M. K. Ieong, P. M. Solomon, S. E. Laux, H. S. P. Wong, and D. Chidambarrao, “Comparison of Raised and Schottky Source/Drain MOSFETs Using a Novel Tunneling Contact Model,” *IEDM*, pp. 733–736, 1998.

[129] K. S. Kundert, J. K. White, and A. Sangiovanni-Vincentelli, *Steady State Methods for Simulating Analog and Microwave Circuits*, Kluver Academic Publishers, 1990.

[130] Y. Takahashi, K. Kunihiro, and Y. Ohno, *IEICE Trans. on Electronics*, vol. E82-C, no. 6, pp. 917–923, 1999.

[131] L. A. Coldren and S. W. Corzine, *Diode Lasers and Photonic Integrated Circuits*, Wiley Series, 1995.

[132] S. L. Chuang, *Physics Of Optoelectronic Devices*, Wiley Series, 1995.

[133] M. Grupen and K. Hess, “Simulation of Carrier Transport and Nonlinearities in Quantum Well Laser Diodes,” *IEEE Journal of Quantum Electronics*, vol. 34, no. 1, pp. 120–140, January, 1998.

[134] C. H. Henry, “Theory of Spontaneous Emission Noise in Open Resonators and its Application to Lasers and Optical Amplifiers,” *Journal of Lightwave Technology*, vol. 4, no. 3, pp. 288–297, March, 1986.

[135] M. Koshiba, *Optical Waveguides by the Finite Element Method*, KTK Scientific Publishers, 1992.

[136] G. Sleijpen, A. Booten, D. Fokkema, and H. van der Vorst, “Jacobi–Davidson Type Methods for Generalized Eigenproblems and Polynomial Eigenproblems,” *Preprint 923*, September, 1995.

[137] D. Fokkema, G. Sleijpen, and H. van der Vorst, “Jacobi–Davidson Type QR and QZ Algorithms for the Reduction of Matrix Pencils,” *SIAM J. Sci. Comput.* vol. 20, pp. 94–125, August, 1998.

[138] Z. M. Li, M. Dion, S. P. McAlister, R. L. Williams, and G. C. Aers, “Incorporation of Strain Into a Two–Dimensional Model of Quantum–Well Semiconductor Lasers,” *IEEE Journal of Quantum Electronics*, vol. 29, no. 2, pp. 346–354, February, 1993.

[139] C. A. Hougen, *Journal of Applied Physics*, vol. 66, p. 3763, 1989.

[140] J. Bardeen and W. Shockley, “Deformation potentials and mobilities in non-planar crystals,” *Phys. Rev.*, vol. 80, p. 72, 1950.

[141] I. Goroff and L. Kleiman, “Deformation potentials in silicon. III. Effects of general strain on conduction and valence levels,” *Phys. Rev.*, vol. 132, p. 1080, 1963.

[142] J. J. Wortman, J. R. Hauser, and R. M. Burger, “Effect of mechanical stress on pn-junction device characteristics,” *Journal of Applied Physics*, vol. 35, p. 2122, 1964.

[143] P. Smeys, *Geometry and stress effects in scaled integrated circuit isolation technologies*, Ph.D. thesis, Department of Electrical Engineering, Stanford University, 1996.

[144] S. Reggiani, M. Valdinoci, L. Colalongo, and G. Baccarani, “A unified analytical model for bulk and surface mobility in Si n- and p-channel MOSFETs,” *ESSDERC*, 1999.

[145] D.E.I.S.- University of Bologna: ESPRIT Project 23643 ESDEM - Project Deliverable 1.1.3.5, 1999.

[146] S. Takagi, A. Toriumi, M. Iwase, and H. Tango, “On the universality of inversion layer mobility in Si MOSFETs: part I - Effects of substrate impurity concentration,” *IEEE Transactions on Electron Devices*, vol. ED-41, no. 12, pp. 2357–2362, 1994.

[147] “A unified mobility model for numerical simulation,” PARASITICS Report, D.E.I.S.-University of Bologna, 1999.

[148] M. Valdinoci, D. Ventura, M.C. Vecchi, M. Rudan, G. Baccarani, F. Illien, A. Stricker, and L. Zullino, “Impact ionization in silicon at large operating temperature,” *SISPAD Proc.*, pp. 27–30, 1999.

[149] M. C. Vecchi and M. Rudan, *IEEE Transactions on Electron Devices*, vol. ED-45, no. 1, pp. 230–238, 1998.

[150] M. G. Ancona and H. F. Tiersten, “Macroscopic physics of the silicon inversion layer,” *Phys. Rev. B*, vol. 35, no. 15, pp. 7959–7965, May, 1987.

[151] M. G. Ancona and G. J. Iafrate, “Quantum correction to the equation of state of an electron gas in a semiconductor,” *Phys. Rev. B*, vol. 39, no. 13, pp. 9536–9540, May, 1989.

[152] A. Wettstein, *Quantum effects in MOS devices*, Ph.D. thesis, ETH Zürich, 2000.

[153] L. Colalongo, M. Valdinoci, G. Baccarani, P. Migliorato, G. Tallarida, and C. Reita, “Numerical Analysis of Poly-TFTs Under Off Conditions,” *Solid-State Electronics*, vol. 41, no. 4, pp. 627–633, 1997.

[154] E. Kapon, *Semiconductor Lasers 2, Materials and Structures*, AP Academic Press, 1999.

[155] K. Rajkanan, R. Singh, and J. Shewchun, “Absorption Coefficient Of Silicon For Solar Cell Calculations,” *Solid-State Electronics*, vol. 22, pp. 793–795, 1979.

[156] K. Kells, *General Electrothermal Semiconductor Device Simulation*, Konstanz: Hartung-Gorre, 1994.

[157] C. K. Williams, T. H. Glisson, J. R. Hauser, and M. A. Littlejohn, “Energy bandgap and lattice constant contours of III-V quaternary alloys of the form $A_xB_yC_zD$ or $AB_xC_yD_z$,” *Journal of Electronic Materials*, vol. 7, no. 5, pp. 639–646, 1978.

[158] I. Vurgaftman, J. R. Meyer, and L. R. Ram-Mohan, “Band parameters for III-V compound semiconductors and their alloys,” *Journal of Applied Physics*, vol. 89, no. 11, pp. 5815–5875, 2001.

[159] T. H. Glisson, J. R. Hauser, M. A. Littlejohn, and C. K. Williams, “Energy bandgap and lattice constant contours of III-V quaternary alloys,” *Journal of Electronic Materials*, vol. 7, no. 1, pp. 1–16, 1978.

[160] M. P. C. M. Krijn, “Heterojunction band offsets and effective masses in III-V quaternary alloys,” *Semicond. Sci. Technol.*, vol. 6 , pp. 27–31, 1991.

[161] S. Adachi, “Band gaps and refractive indices of AlGaAsSb, GaInAsSb, and InPAsSb: Key properties for a variety of the 2–4-µm optoelectronic device applications,” *Journal of Applied Physics*, vol. 61, pp. 4869–4876, 1987.

[162] R. L. Moon, G. A. Antypas, and L. W. James, “Bandgap and lattice constant of GaInAsP as a function of alloy composition,” *Journal of Electronic Materials*, vol. 3, no. 3, pp. 635–644, 1974.

[163] B. Schmithüsen, *Grid Adaptation for the Stationary Two-Dimensional Drift-Diffusion Model in Semiconductor Device Simulation*, Ph.D. thesis, ETH Zurich, 2002.

[164] B. Schmithüsen, K. Gärtner, and W. Fichtner, “A Grid Adaptation Procedure for the Stationary 2D Drift-Diffusion Model Based on Local Dissipation Rate Error Estimation: I - Background, II- Examples,” Technical Reports 2001/02 and 2001/03, Integrated Systems Laboratory, ETH Zurich, Switzerland, 2001.

[165] R. Verfürth, *A Review of A Posteriori Error Estimation and Adaptive Mesh-Refinement Techniques*, Wiley-Teubner, 1996.

[166] A. Plonka, “Time-dependent Reactivity of Species in Condensed Media,” *Lecture Notes in Chemistry*, New York: Springer-Verlag, 1986.

[167] C. Hu et al., “Hot-electron-induced MOSFET degradation-model, monitor, improvements,” *IEEE ED*, vol. 32, no. 2, pp. 375–385, 1985.

[168] K. Hess et al., “Theory of channel hot-carrier degradation in MOSFETs,” *Physical Review B*, vol. 272, pp. 527–531, 1999.

[169] B. Tuttle et al., “Structure, energetics, and vibration properties of Si-H bond dissociation in silicon,” *Physical Review B*, vol. 59, no. 20, pp. 12884–12889, 1999.

[170] Z. Chen et al., “On the Mechanism for Interface Trap Generation in MOS Transistors Due to Channel Hot Carrier Stressing,” *IEEE EDL*, vol. 21, no. 1, pp. 24–36, 2000.

[171] O. Penzin, A. Haggag, W. McMahon, E. Lyumkis, and K. Hess, “MOSFET Degradation Kinetics and Simulation,” submitted to IEEE Transaction on ED.

[172] J. W. McPherson et al., “Complementary model for intrinsic time-dependent dielectric breakdown in SiO2 dielectrics,” *Journal of Applied Physics*, vol. 88, no. 9, pp. 5351–5359, 2000.

[173] K. Varahramyan and E. J. Verret, “A model for specific contact resistance applicable for titanium silicide-silicon contacts,” *Solid-State Electronics*, vol. 39, no. 11, pp. 1601–1607, 1996.

[174] F. M. Bufler and W. Fichtner, *Applied Physics Letters*, vol. 81, no. 1, pp. 82–84, 2002.

[175] M. T. Currie et al., *Journal of Vacuum Science & Technology B*, vol. 19, no. 6, pp. 2268–2279, 2001.

[176] C. W. Leitz et al., *Journal of Applied Physics*, vol. 92, no. 7, pp. 3745–375, 2002.

[177] L. F. Register, E. Rosenbaum, and K. Yang, “Analytic model for direct tunneling current in polycrystalline silicon-gate metal–oxide–semiconductor devices,” *Applied Physics Letters*, vol. 74, no. 3, pp. 457–459, 1999.

[178] S. K. Banerjee et al., “Compact Modeling of MOSFETs with High-K Gate Dielectrics,” unpublished.

[179] H. Matsuura, *International Conference on SiC and Related Materials - ICSCRM2001,* Tsukuba, Japan 2001.

[180] Y. P. Yu and M. Cardona, *Fundamentals of Semiconductors: Physics and Materials Properties,* Chapter 4, Berlin: Springer, 2nd ed., 1999.

[181] K. F. Brennan, *The Physics of Semiconductors with Application to Optoelectronic Devices*, Chapter 5, Cambridge: Cambridge University Press, 1999.

[182] J. L. Egley and D. Chidambaro, “Strain Effect on Device Characteristics: Implementation in Drift-Diffusion Simulators,” *Solid-State Electronics*, vol. 36, no. 12, pp. 1653–1664, 1993.

[183] G. L. Bir and G. E. Pikus, *Symmetry and Strain-Induced Effects in Semiconductors*, New York: Wiley, 1974.

[184] R. Smith, M. Alam, G. Baraff, and M. Hybertsen, “Numerical methods for semiconductor laser simulations,” *Proc. IWCE*, 1997.

[185] M. Streiff, A. Witzig, M. Pfeiffer, P. Royo, and W. Fichtner, “A comprehensive VCSEL device simulator,” to be published in *IEEE Journal of Selected Top. Quantum Electronics*, 2003.

[186] G. A. Baraff and R. K. Smith, “Nonadiabatic semiconductor laser rate equations for the large-signal, rapid-modulation regime,” *Physical Review A*, vol. 61, pp. 043808-1–043808-13, 2000.

[187] J. M. Jin, *The Finite Element Method in Electromagnetics*, Wiley-Interscience, 2nd ed., 2002.

[188] J.-P. Berenger, “A perfectly matched layer for the absorption of electromagnetic waves,” *J. Comput. Phys.*, vol. 114, no. 2, pp. 185–200, 1994.

[189] W. C. Chew and W. H. Weedon, “A 3D perfectly matched medium from modified Maxwell’s equations with stretched coordinates,” *Microwave Opt. Tech. Letters*, vol. 7, pp. 599–604, 1994.

[190] F. L. Teixeira and W. C. Chew, “Systematic derivation of anisotropic PML absorbing media in cylindrical and spherical coordinates,” *IEEE Microwave and Guided Wave Letters*, vol. 7, no. 11, pp. 371–373, 1997.

[191] F. L. Teixeira and W. C. Chew, “A general approach to extend Berenger’s absorbing boundary condition to anisotropic and dispersive media,” *IEEE Transactions on Antennas and Propagation*, vol. 46, no. 9, pp. 1386–1387, 1998.

[192] M. Ahmed and M. Yamada, “An infinite order perturbation approach to gain calculation in injection semiconductor lasers,” *Journal of Applied Physics*, vol. 84, no. 6, pp. 3004–3015, 1998.

[193] B. Witzigmann, A. Witzig, and W. Fichtner, “Nonlinear gain saturation for 2-dimensional laser simulation,” *Proceedings of LEOs Conference*, pp. 659–660, 1999.

[194] R. Eppenga, M. F. H. Schuurmans, and S. Colak, “New k.p theory for GaAs/Ga(1-x)Al(x)As-type quantum wells,” *Phy. Rev. B*, vol. 36, no. 3, pp. 1554–1564, 1987.

[195] A. Witzig, L. Schneider, M. Pfeiffer, M. Streiff, T. Lundstroem, P. Tikhomirov, W.-C. Ng, and W. Fichtner, “Optimization of a leaky-waveguide laser using DESSIS,” *Proceedings of the IEEE/LEOs 3rd International Conference on Numerical Sim. of Semicon. Optoelectron. Dev.*, NUSOD-03, Tokyo, Japan, pp. 35–36, October 2003.

[196] S. E. Laux, “Application of Sinusoidal Steady-State Analysis to Numerical Device Simulation,” in *New Problems and New Solutions for Device and Process Modelling*, Dublin: Boole Press, pp. 60–71, 1985.

[197] A. Schenk, *Advanced Physics Models for Silicon Device Simulation*, Wien: Springer-Verlag, 1998.

[198] S. L. Chuang and C. S. Chang, “A band-structure model of strained quantum-well wurtzite semiconductors,” *Semiconductor Science and Technology*, vol. 12, no. 3, pp. 252–263, 1997.