Sentaurus™ Device QTX User Guide

Version T-2022.03, March 2022

SYNOPSYS®

Copyright and Proprietary Information Notice

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

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

Third-Party Links

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

www.synopsys.com

Contents

About This Guide

This user guide describes the Synopsys® Sentaurus™ Device QTX tool that features the subband-based Boltzmann transport equation solver (Subband-BTE solver) and the quantum transport equation solver (NEGF solver).

This user guide is intended for users of TCAD Sentaurus involved in advanced CMOS transport modeling.

For additional information, see:

- The TCAD Sentaurus release notes, available on the Synopsys SolvNetPlus support site (see Accessing SolvNetPlus)
- Documentation available on the SolvNetPlus support site

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`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.

Customer Support

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

Accessing SolvNetPlus

The SolvNetPlus support site includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The site also gives you

access to a wide range of Synopsys online services, which include downloading software, viewing documentation, and entering a call to the Support Center.

To access the SolvNetPlus site:

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

Contacting Synopsys Support

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

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

Contacting Your Local TCAD Support Team Directly

Send an email message to:

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

Acknowledgments

ILS was codeveloped by Integrated Systems Laboratory of ETH Zurich in the joint research project NUMERIK II with financial support by the Swiss funding agency CTI.

Part I: Subband-BTE Solver

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

1

Introduction to the Subband-BTE Solver

This chapter describes the features of the subband-based Boltzmann transport equation solver of Sentaurus Device QTX and the physical models it uses to model quasiballistic transport.

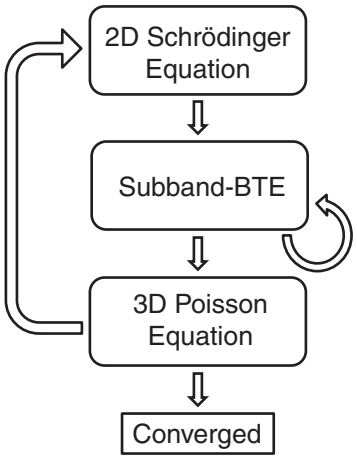
Features of the Subband-BTE Solver

Sentaurus Device QTX features the subband-based Boltzmann transport equation solver (hereafter, referred to as the *Subband-BTE solver*) that takes into consideration the effects of strong quantum confinement as well as quasiballistic transport along the channel of small FinFETs and nanowires. Both purely ballistic transport and scattering due to nonpolar phonon, polar optical phonon, surface roughness, alloy, and Coulomb scattering can be treated.

The subbands used by the Subband-BTE solver are provided by the solution of the Schrödinger equation on 2D slices along the channel. The solution of the Subband-BTE solver provides the distribution functions of carriers in each subband. This information, combined with the subband dispersion and the wavefunctions from the Schrödinger solver, provides the carrier density used in the Poisson equation. As shown in Figure 1 on page 12, at each bias point, the Schrödinger equation, the subband-based Boltzmann transport equation (subband-BTE), and the Poisson equation are iterated until convergence.

The main output of the solution of the subband-BTE is the current flowing through the channel. In addition, several internal quantities related to the solution of the subband-BTE, such as the distribution function, the current spectrum, and the carrier velocity, can be saved to TDR files.

Figure 1 *Schematic of the iteration procedure for the self-consistent solution of the Schrödinger equation, the subband-BTE, and the Poisson equation*



Similarities to Sentaurus Band Structure

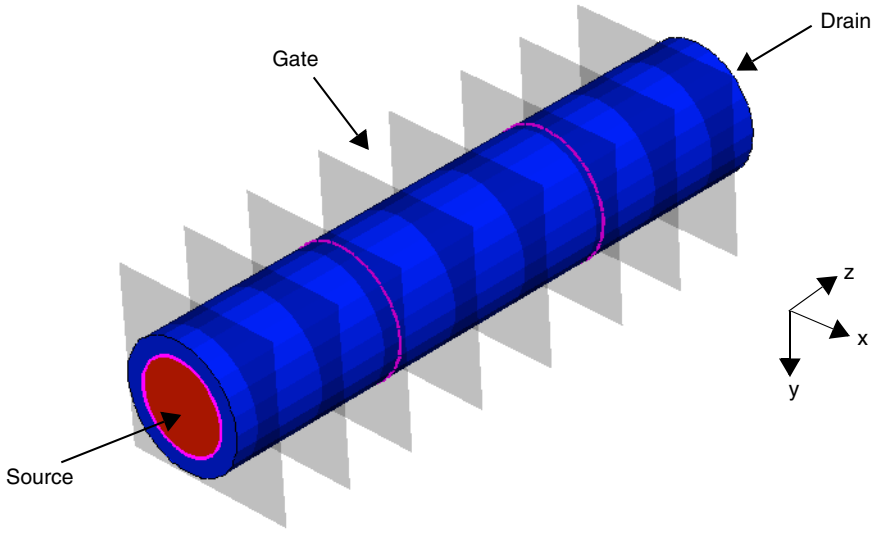
The Subband-BTE solver shares many commands and physical models with Sentaurus Band Structure.

For descriptions of common commands, models, and parameters, see Part II of the *Sentaurus™ Device Monte Carlo User Guide*.

Device Structure

The simulation of carrier transport using the solution of the subband-BTE can be performed only for 3D devices. The use of the Subband-BTE solver is most appropriate for devices with a well-defined channel, in which geometric quantum confinement plays a strong role. This is typically the case in small FinFETs and nanowire devices. The device structure is loaded from a TDR file using the `LoadDevice` command.

Figure 2 Three-dimensional nanowire device showing slices and contacts



Requirements and Restrictions on Device Structure and Mesh

The Subband-BTE solver has some requirements and restrictions on the type of device structure and mesh that can be treated, namely:

- The 3D mesh must be composed of all tetrahedra or cuboids.
- In standalone Sentaurus Device QTX simulations, the 3D mesh can be either an extruded mesh from a 2D cross section or an unstructured Delaunay mesh that contains a set of well-defined planes of mesh points perpendicular to the z-axis. For automated slice extraction along the channel direction, all mesh points should fall on the slice locations, which are typically found only in extruded structures. With an unstructured mesh, you must specify the slice locations.
- In coupled Sentaurus Device–Sentaurus Device QTX simulations, the 3D mesh is typically an unstructured Delaunay mesh that, like for the standalone mode, must contain several well-defined planes of mesh points perpendicular to the z-axis. This simulation mode allows the use of the Subband-BTE solver on a subdomain of the structure, such as the channel region of a FinFET. See Using Subband-BTE Solver as an External Solver for Sentaurus Device on page 16 and Chapter 3 on page 39.

Note:

When using an unstructured Delaunay mesh, the device mesh must contain a set of well-defined planes. This often requires modifying the Sentaurus Process command file or including an additional tool instance of Sentaurus Mesh, as

discussed in Chapter 3 on page 39. The locations of these planes, as enforced in the additional Sentaurus Mesh tool instance, can then be input to Sentaurus Device QTX.

Device Axes and Orientation

The device axis along the channel direction must be the z-axis. It must be set up correctly during structure generation. The confinement axes must be the x-axis and the y-axis. The orientation of the device relative to the crystallographic axes can be specified using the `xDirection` and `yDirection` arguments of the `Physics` command.

Contacts

It is expected that the left and right ends of the structure are terminated by contacts, as shown in Figure 2 on page 13. Additional contacts, typically one gate contact, can be included in the structure as well. However, the current is computed only for the left and right contacts of a structure.

During the solution of the Poisson equation, a special Neumann boundary condition should be used at the left and right contacts to allow charge neutrality there to be maintained, even in the presence of quantum confinement. To activate this boundary condition, use the `Physics contact` command with `type=Neumann`. For example:

```
Physics contact=source type=Neumann
Physics contact=drain  type=Neumann
```

You can specify a lumped resistance, in units of ohm (Ω), for the left and right contacts of a sliced channel. For example, to specify a lumped resistance of 1 kΩ for contacts named `source` and `drain`, specify:

```
Physics contact=source resist=1.0e3
Physics contact=drain  resist=1.0e3
```

Sliced Channels

To solve the subband-BTE along the channel of a device, the subbands from the solution of the 2D Schrödinger equation at slices along the channel must be computed first. A group of these slices forms a *sliced channel*. On each slice, a 2D Schrödinger equation is solved for the subband dispersions and wavefunctions. Based on these quantities, along with the distribution function for each subband, the carrier density on each node in the slice can be computed. Then, this is interpolated on to the 3D device mesh for solving the 3D Poisson equation.

Creating a Sliced Channel

You create a sliced channel by using the `Math slicedChannel` command (see Math slicedChannel on page 45). This is similar to the `Math nonlocal` command used in other parts of Sentaurus Band Structure. The parts of the device structure to be included in the sliced channel can be specified by using either a list of regions, or a bounding box, or both.

When simulating an extruded structure or one in which all mesh points fall on slice locations, to create a sliced channel consisting of only the silicon region, named `si1`, of a silicon nanowire, use the following command:

```
Math slicedChannel name=channel1 regions=[list si1]
```

During the creation of a sliced channel using this `Math` command, 2D slices along the channel are extracted automatically.

When simulating a structure that has well-defined planes perpendicular to the z-axis and that might have additional mesh points located between these planes, the creation of the sliced channel must be modified to include the locations of the planes. This is typically the case when simulating with an unstructured Delaunay mesh.

For example, to include slice locations positioned at 1 nm intervals over a 30 nm long device, consisting of only the silicon region, named `si1`, of a silicon nanowire, use the following commands:

```
# Specify coordinate list of the mesh planes created during structure
# generation used to extract slices from the 3D device (unit: um)

set sliceCoordList [list 0.0 0.001 0.002 ... 0.029 0.03]

Math slicedChannel name=channel1 regions=[list si1] \
   sliceCoords=$sliceCoordList
```

In both cases described here, a nonlocal area is created automatically on each slice with a name consisting of the sliced channel `name` and the suffix `.NL`, for example, `channel1.NL`.

Specifying a Schrödinger Solver for a Sliced Channel

Specifying a Schrödinger solver for a sliced channel is very similar to specifying a Schrödinger solver for a nonlocal area for mobility calculations in Sentaurus Band Structure. For a sliced channel, the `Physics slicedChannel` command is used, and the specified Schrödinger solver is used for all slices in the sliced channel (see Physics slicedChannel on page 47).

Note:

You can use dynamic subband numbers here to select the optimal number of subbands automatically (see *Sentaurus™ Device Monte Carlo User Guide*, Dynamically Varying Subband Numbers).

Examples

```
Physics slicedChannel=channel1 eSchrodinger=Parabolic Nsubbands=8 \
   Nk=16 valleys=[list Delta1 Delta2 Delta3] Kmax=0.6 a0=5.43e-8 \
   correction=3

Physics slicedChannel=channel1 hSchrodinger=6kp \
   valleys=[list Gamma] Nsubbands=32 Nk=16 Kmax=0.6 a0=5.43e-8
```

Using Subband-BTE Solver as an External Solver for Sentaurus Device

Similar to the existing setup for Sentaurus Device and Sentaurus Band Structure, you can run coupled Sentaurus Device and Sentaurus Device QTX simulations of 3D structures with 2D confinement.

This simulation scheme allows coupled carrier–Poisson simulations in Sentaurus Device where, on a subdomain of the structure, the quantum-correction potential and effective mobility are extracted from the Subband-BTE solver. For each Newton step of the coupled carrier–Poisson equation system, the subband-BTE system is solved on the subdomain. The Subband-BTE solver uses the electrostatic potential provided by Sentaurus Device as input and provides the quantum-correction potential and effective mobility as output to Sentaurus Device.

For the necessary modifications to the structure creation to activate this simulation mode, see Device Structure on page 39.

The minimal modifications needed to a standalone Sentaurus Device QTX command file are described in Creating a Sliced Channel on page 15, referring to the use of an unstructured Delaunay mesh, and Performing the $I_{d–Vg}$ Simulation on page 40.

See *Sentaurus™ Device User Guide*, External Boltzmann Solver, for the required modifications to the Sentaurus Device command file.

Model and Parameter Specification

Similar to the specification of models and parameters for the subband and mobility calculations in Sentaurus Band Structure, for the Subband-BTE solver, you can specify models and parameters on a material or region basis.

The specified models and parameters are used on all slices that contain the specified material or region. The following sections highlight the model types that are particularly important for the Subband-BTE solver: valley models and scattering models.

Valley Models

Valley models specify the band model and parameters to use when solving the Schrödinger equation. The following valley models can be used for the subband-BTE:

- The `ConstantEllipsoid`, `2kpEllipsoidal`, and `2kp` models can be used for the description of electrons.
- The `6kp` valley model gives an accurate description of holes.
- The `3kp` valley model is an approximation of `6kp`, where the split-off band is not captured due to the neglect of spin-orbit coupling.

Scattering Models

Scattering models are used to model specific, microscopic scattering transitions such as phonon and surface roughness scattering. All of the scattering models that can be used for mobility calculations in Sentaurus Band Structure on 2D structures can also be used when solving the subband-BTE. These models include elastic and inelastic nonpolar phonon scattering, polar optical phonon scattering, alloy scattering, surface roughness scattering, and Coulomb scattering from bulk impurities, interface charge, and traps. In addition, the surface roughness and the Coulomb scattering models can be screened using either the scalar or tensor dielectric function models.

A standard set of scattering models is set up for silicon regions by default, as described in the documentation of Sentaurus Band Structure in the *Sentaurus™ Device Monte Carlo User Guide*.

For purely ballistic simulations, all scattering models must be removed. To remove all scattering models, use the following command:

```
Physics material=all ScatteringModel removeAll
```

RTA Scattering Model

The relaxation time approximation (RTA) scattering model is available for solving the subband-BTE. This model is implemented as an intra-subband scattering mechanism using the transition rate given in [1]:

$$S_{\mathrm{RTA}} = \frac{f_n(z,\varepsilon) - f_n^{\mathrm{leq}}(z,\varepsilon)}{\tau} \tag{1}$$

where ε is the total energy and f_n^{leq} is an equilibrium Fermi–Dirac distribution function defined in terms of a local Fermi level, E_F^{leq}:

$$f_n^{\text{leq}}(z, \varepsilon) = \frac{1}{1 + e^{(\varepsilon - E_F^{\text{leq}})/k_B T}} \tag{2}$$

where T is the ambient temperature. The only adjustable parameter in the RTA scattering model is τ, the relaxation time which is taken as a constant. The local Fermi level, E_F^{leq}, is determined by enforcing a carrier conservation condition on the equilibrium Fermi–Dirac distribution function integrated over energy. This produces an additional equation that must be solved along with the subband-BTE.

The local Fermi level can be saved to a TDR file for visualization using the `BTESaveCC` command (see BTESaveCC on page 56).

The RTA scattering model serves the following purposes:

- It can stabilize the numeric solution of the subband-BTE. By default, an RTA scattering model is always used with a built-in τ = 1e-10 s, which is a long enough relaxation time so as not to impact the computed current and is a short enough relaxation time so as to provide efficient numeric stability. You can change the value of τ for the RTA scattering model when selecting the numeric Subband-BTE solver as described next (see Equation 3).
- It provides an efficient model that can mimic more complicated mechanisms that might incur significant runtime or are not yet available. For this purpose, you can specify additional RTA scattering models using the `Physics ScatteringModel` command on a per-region and per-valley basis. For example:

```
Physics region=si1 ScatteringModel=RTA name=rta1 \
   valleys=[list Delta1] tau=1.0e-13
```

- The final value of τ that is used in the subband-BTE is computed using Mathiessen's rule considering the built-in RTA scattering model and all user-defined RTA scattering models for a particular region and valley, that is:

$$\frac{1}{\tau} = \frac{1}{\tau_{\text{built-in}}} + \sum_i \frac{1}{\tau_i} \tag{3}$$

Surface Roughness

Surface roughness scattering is activated by specifying the `SRFor2D` scattering model. This model is used because it is applied to 2D slices. The `SRFor2D` model applies surface roughness scattering only at semiconductor–insulator interfaces.

During the creation of the sliced channel, the Subband-BTE solver automatically extracts any insulator regions that are adjacent to the specified semiconductor regions.

This does not affect the solution of the Schrödinger equation on the sliced channel, but allows the `SRfor2D` model to work correctly even if no insulator regions are specified explicitly.

Coulomb Scattering

When used with the Subband-BTE solver, Coulomb scattering can be applied to both bulk doping and interface charge.

To speed up the calculation of the matrix elements for Coulomb scattering, the `whenToCompute` parameter can be used to specify when the matrix elements should be recomputed. The options for this parameter are:

- The `EveryIteration` option causes the matrix elements to be evaluated for each Poisson Newton iteration. This option provides the best accuracy but requires the most runtime. This is the default value.
- The `FirstIteration` option causes the matrix elements to be evaluated only for the first Poisson Newton iteration during a `Solve` command. Essentially, the solution from the previous bias point is used to compute the matrix elements. For subsequent Newton iterations, the matrix elements are reused when computing the scattering rate. This approach can potentially affect accuracy, but it greatly reduces the runtime. For bias ramps, this approach has worked well. If accuracy is important, a bias point can be repeated.

The following example shows how to specify the `whenToCompute` parameter:

```
Physics material=Silicon ScatteringModel=Coulomb name=eCoulomb \
   transitionType=Intravalley valleys=[list Delta1 Delta2 Delta3] \
   screening=Lindhard whenToCompute=FirstIteration
```

Screening

Some of the scattering models, such as surface roughness and Coulomb, can be screened using one of the available screening models. By default, no screening is used. The required screening model can be selected using the `screening` parameter for each scattering model.

This parameter can be set to one of the following values:

- `none` for no screening
- `Lindhard` for the scalar Lindhard screening model
- `LindhardTDF` for the tensor Lindhard screening model

The screening models utilize the polarization (Π), which characterizes the strength of the screening, and the dielectric form-factor (F), which characterizes the screening effectiveness of each subband pair in terms of the wavefunction overlap with the Coulomb Green's function.

The polarization is given by:

$$\Pi_{nn'}(q) = \frac{g_s \cdot g_v}{2\pi}\int dk \frac{f_{n'}(k+q) - f_n(k)}{E_n(k) - E_{n'}(k+q)} \tag{4}$$

where:

- g_s and g_v are the spin and valley degeneracy, respectively.
- E_n and f_n are the dispersion and distribution function for subband n, respectively.

The dielectric form-factor is given by:

$$F^{nn'}_{mm'}(q) = \iint d\boldsymbol{x} d\boldsymbol{x}' \ \psi_m(\boldsymbol{x}) \cdot \psi_{m'}{}^{\dagger}(\boldsymbol{x}) G(q, \boldsymbol{x}, \boldsymbol{x}') \psi_n{}^{\dagger}(\boldsymbol{x}') \cdot \psi_{n'}(\boldsymbol{x}') \tag{5}$$

where Ψ_m is the wavefunction for subband m, and G is the Coulomb Green's function.

Tensor Dielectric Function

The tensor dielectric function (TDF) in the TDF screening model is given by:

$$\varepsilon^{nn'}_{mm'}(q) = \delta_{mn}\delta_{m'n'} + \Pi_{nn'}(q) F^{nn'}_{mm'}(q) \tag{6}$$

The `tensor Lindhard screening model (LindhardTDF)` is time consuming to compute, particularly when a large number of subband pairs is used for the screening calculation. The number of subband pairs can be controlled using the `tdfDegenTol` parameter of the `Physics slicedChannel` command when specifying the Subband-BTE solver. Subband pairs are used in the screening calculation if their subband edges are within `tdfDegenTol` of each other. It is recommended to set this to a value of 1.0e-4 eV.

Scalar Dielectric Function

The scalar dielectric function (SDF) in the SDF screening model is given by:

$$\varepsilon(q) = 1 + \sum_n \Pi_{nn}(q) F^{nn}_{nn}(q) \tag{7}$$

The sum is over the subbands. While it is expensive to compute, the SDF screening model is much faster than the TDF screening model and provides a good trade-off between accuracy and runtime.

Runtime Speedup With Screening

The screening models are computationally expensive. They are heavily parallelized and you should use as many threads as possible. However, even with many threads, for a structure with many slices, the overall wallclock time when using screening might still be quite long.

To provide a trade-off between accuracy and runtime, when specifying the Subband-BTE solver, use the `whenToComputeScreening` argument of the `Physics slicedChannel` command to control how often the screening model is evaluated:

- If `whenToComputeScreening=EveryIteration`, the screening model is evaluated for each Poisson Newton iteration. This option provides the best accuracy but requires the most runtime. This is the default.
- If `whenToComputeScreening=FirstIteration`, the screening model is evaluated only for the first Poisson Newton iteration during a `Solve` command. Essentially, the solution from the previous bias point is used to compute the screening dielectric function. For subsequent Newton iterations, the dielectric function is reused. This approach can potentially affect accuracy, but it greatly reduces the runtime. For bias ramps, this approach has worked well. If accuracy is important, a bias point can be repeated.

The following example shows how to specify the `whenToComputeScreening` parameter and the `tdfDegenTol` parameter:

```
Physics slicedChannel=channel1 eBTESolver=Numeric \
   tdfDegenTol=1.0e-4 whenToComputeScreening=FirstIteration
```

Subband-Based Boltzmann Transport Equation

With quantum confinement in two dimensions, the subband-BTE is reduced to one dimension along the channel within each subband. The solution of the subband-BTE gives the distribution function as a function of the *k*-vector and the channel coordinate for each subband.

Using *z* for the channel axis, for subband *n*, the subband-BTE is [2]:

$$-\frac{\partial f_n}{\partial k}\frac{1}{\hbar}\frac{\partial E_n}{\partial z}+\frac{\partial f_n}{\partial z}\frac{1}{\hbar}\frac{\partial E_n}{\partial k}=S_{n,\,\mathrm{in}}-S_{n,\,\mathrm{out}} \tag{8}$$

where:

- f_n is the distribution function for subband *n*.
- E_n is the dispersion for subband *n*.

With Fermi–Dirac statistics, the in-scattering term ($S_{n,\mathrm{in}}$) and the out-scattering term ($S_{n,\mathrm{out}}$) are given by:

$$\begin{aligned} S_{n,\,\mathrm{in}} &= \sum_{n'}\frac{1}{2\pi}\int dk' S_{n'n}(k',k) f_{n'}(z,k')[1-f_n(z,k)] \\ S_{n,\,\mathrm{out}} &= \sum_{n'}\frac{1}{2\pi}\int dk' S_{nn'}(k,k') f_n(z,k)[1-f_{n'}(z,k')] \end{aligned} \tag{9}$$

where $S_{n'n}(k',k)$ is the total transition rate due to scattering.

As boundary conditions on the distribution functions, equilibrium Fermi–Dirac distribution functions at the left and right contacts are applied to carriers being injected into the channel.

Selecting a Subband-BTE Solver

The following Subband-BTE solvers are available for solving the subband-BTE:

- `AnalyticBallistic` is the faster and simpler solver. As its name suggests, it can be used only for purely ballistic simulations. This solver uses the analytic solution to the subband-BTE, which is determined by projecting the injecting, equilibrium Fermi–Dirac distribution functions from the contacts throughout the device as determined by the top of each subband barrier.
- `Numeric` performs an actual numeric solution to the subband-BTE. This solver can be used for purely ballistic transport as well as with all of the available scattering models including the RTA scattering model.

For example, to activate the `AnalyticBallistic` subband-BTE for electron transport for a sliced channel named `channel1`, you can use the following command:

```
Physics slicedChannel=channel1 eBTESolver=AnalyticBallistic
```

For a subband-BTE simulation of holes, you can use the `hBTESolver` argument:

```
Physics slicedChannel=channel1 hBTESolver=AnalyticBallistic
```

Energy Grid

The subband-BTE is solved on a 2D tensor-product grid. One axis of this grid is along the channel direction and is referred to as the *channel coordinate axis*. The grid points along this axis are given by the slice locations along the channel. The other axis of the BTE solution grid is the *total energy axis*. Here, *total energy* refers to the total energy of a carrier in a particular subband as determined by the sum of the minimum subband energy and the kinetic energy of the carrier.

The minimum and maximum energies used in the energy grid are determined automatically. The nominal energy grid spacing is set using the `deltaE` argument of the `Physics slicedChannel` command when selecting a Subband-BTE solver (see Physics slicedChannel on page 47).

In addition, a special refinement around the top of each subband can be activated to improve the accuracy of the calculation and to improve convergence. This is activated using the `resolveTOBEnergy` argument. The smallest energy spacing used for this refinement is set using the `minDeltaE` argument.

For example, a typical energy grid setup for a sliced channel named `channel1` is:

```
Physics slicedChannel=channel1 eBTESolver=Numeric deltaE=4e-3 \
   resolveTOBEnergy=1 minDeltaE=1.0e-4
```

Numeric Options

When specifying the Subband-BTE solver, you can set several arguments related to numeric details, in particular:

- `approximateJacobian=`*`Boolean`*: When true (1), this argument activates a special algorithm to approximate the Jacobian that is used when solving the BTE system. This can greatly reduce memory consumption with a small impact on convergence. Default: 1
- `NkForScreening=`*`Integer`*: This argument sets the number of *k*-points to use when evaluating screening. Default: 32
- `useParabolicFitForToB=`*`Boolean`*: When true (1), this argument activates a special algorithm to more accurately locate the top of the source–drain barrier. This mainly applies to FET devices. Default: 0
- `useKdependentWFForBTEDensity=`*`Boolean`*: When true (1), this argument computes the carrier density based on *k*-dependent wavefunctions. When this argument is false (0), Γ-point wavefunctions are used, which improves convergence. Default: 0

  Note:

  If `useKdependentWFForBTEDensity=0` and `useKdependentWF=1` are set when specifying the Schrödinger solver, then *k*-dependent wavefunctions are still used when computing the scattering rates.

  To have good convergence behavior, if `reorderDispersion=1` is set, then you must also specify `useKdependentWFForBTEDensity=1` and `useKdependentWF=1`.

For details about these arguments, see Physics slicedChannel on page 47.

Optimization Options

With increasing device structure sizes, simulations with the Subband-BTE solver can become computationally heavy both in terms of turnaround time and memory requirements. This is particularly true when using k·p valley models for band structure modeling, for example, the `6kp` valley model is often used for PMOS simulations. The following options are intended to reduce the cost of the Schrödinger eigenvalue problem and the actual Boltzmann transport problem.

Specifying a Simplified Eigensolve

Simulating large structures with the Subband-BTE solver is computationally expensive, and a large proportion of the computational time is spent on the 2D Schrödinger solution. In particular, PMOS simulations using the `6kp` valley model typically require a *k*-dependent solution that further increases the workload.

You can activate a simplified eigensolve that applies a quantum-mechanical perturbation theory approximation to the eigenvalue problem on each slice. This simplified eigensolve is activated when the potential update becomes smaller than a user-defined threshold.

The simplified eigensolve computes an eigenvalue shift that is calculated on a per-subband and per-momenta basis.

Degenerate eigenvalues are captured by defining a tolerance. Any eigenvalues within the tolerance are treated as degenerate.

To set the value of the degenerate eigenvalue tolerance (in eV), use the following argument of the `Physics slicedChannel` command:

```
Physics slicedChannel=String simplifiedEVDegenTol=Double
                                                  # default: 1.e-4
```

In addition, the following argument of the `Math` command allows you to define the potential threshold (in V) for activating the simplified eigensolve:

```
Math potentialThresholdForSimplifiedEV=Double     # default: 0.0 V
```

By default, the simplified eigensolve is switched off.

Note:

For accurate results, take care when specifying the potential threshold value. A good starting point is around 5 mV, but you will most likely need to adjust this value.

Combining Degenerate States

When using the `6kp` or `8kp` valley models, which have an explicit representation of the spin, you can reduce the actual cost of the transport problem by activating the following option:

```
Physics slicedChannel=String combineSpinOfDegenerateStates=1
```

This feature utilizes the degeneracy of the subband dispersion, which is always given in the absence of magnetic fields for the k·p valley models with explicit spin representation, by combining two degenerate states into one state.

Note:

If you activate the `combineSpinOfDegenerateStates` option for valley models other than `6kp` or `8kp`, then an error is generated.

Limiting the Subband Scattering Range

For transition types such as `Intravalley`, `Intervalley`, and `gIntervalley`, a state in the initial subband is allowed, by default, to scatter to all valid final states in all valid final subbands. When using a large number of subbands and a fine energy grid for the solution of the subband-BTE, this produces a very large system matrix that can be very time consuming to solve. Restricting the number of final subbands to which an initial state scatters can help to reduce the overall system size and to improve runtime performance. This can be achieved by using the `subbandScatteringRange` parameter of the `Physics slicedChannel` command (see Physics slicedChannel on page 47).

For details about optimizing screening capabilities, see Runtime Speedup With Screening on page 20. For details about calculating a good initial guess, see Initial Guess on page 27.

Self-Heating

Self-heating effects can have an important role in current or future nanoelectronic devices. Ultra-scaled devices, such as recent transistor structures, exhibit a significantly reduced thermal conductivity compared to bulk values. The combination of low thermal conductivity and power dissipation due to inelastic scattering can lead to significantly increased temperatures (hot spots) in the device, which can further reduce the current or diminish reliability.

Therefore, in the Subband-BTE solver and the NEGF solver, the heat equation can be solved together with the transport problem to include self-heating effects. The stationary heat equation, which is solved in Sentaurus Device QTX, is as follows:

$$\nabla_r \cdot (\kappa(\boldsymbol{r}) \nabla_r T(\boldsymbol{r})) = p_{\text{scattering}}(\boldsymbol{r}) + p_{\text{thContact}}(\boldsymbol{r})\big|_{r \in \text{thContact}} \tag{10}$$

where:

- $\kappa(\boldsymbol{r})$ is the thermal conductivity, given in W/cmK.
- $T(\boldsymbol{r})$ is the temperature, given in K.
- $p_{\text{scattering}}(\boldsymbol{r})$ is the heat source at position $\boldsymbol{r}$ due to inelastic scattering, given in W/cm^3.
- $p_{\text{thContact}}(\boldsymbol{r})$ is the heat source or heat sink if a thermal contact resistance is applied to the corresponding thermal contact, given in W/cm^3.

The dissipated power of inelastic scattering processes serves as a heat source for the heat equation. Furthermore, if a thermal contact resistance is applied to the corresponding thermal contact, then an additional heat source or heat sink term appears. Finally, the computed temperature distribution through the device is fed back to the scattering matrix element calculations.

Example of Self-Heating

First, you must set the appropriate thermal properties of different regions or materials:

```
Physics material=semiconductor \
   ThermalConductivity=ConstantThermalConductivity kappa=0.2
Physics material=insulator \
   ThermalConductivity=ConstantThermal Conductivity kappa=0.02
```

The unit of `kappa` is W/(cmK).

Second, you set the boundary conditions for the heat equation through the `thermalContact` argument of the `Physics` command:

```
Physics thermalContact=source contactTemperature=300 \
   thermalResist=5.0e8
Physics thermalContact=drain  contactTemperature=300 \
   thermalResist=5.0e8
Physics thermalContact=gate   type=Neumann
```

The argument `contactTemperature` defines the temperature [K] of the thermal contacts, and the `type` argument specifies the boundary type, which can be `Dirichlet` (default) or `Neumann`.

To model the thermal environment more realistically, you can also define a thermal resistance by setting `thermalResist` > 0.0 [K/W], which is then connected to the contact. See Physics thermalContact on page 50.

Furthermore, you can change the temperature of the Dirichlet boundaries of the thermal contacts dynamically in the `Solve` command like the voltage values of the contacts. For example:

```
Solve V(source)=0.0   V(drain)=0.7   V(gate)=0.5 \
      T(source)=350.0 T(drain)=350.0 logFile=exampleT_out.plt
```

You must activate the solution of the heat equation through the `Math` command:

```
Math selfHeating=1
```

The argument `selfHeating` specifies whether to allow (`1`) or not allow (`0`) self-heating.

Finally, the computed temperature as well as the extracted heat source due to inelastic scattering can be written in output files. A 3D field of the temperature can be saved to a TDR file by using the `Save` command. For example:

```
Save tdr=Temperature3D_out.tdr models=[list LatticeTemperature \
   HeatSource]
```

You can also save the mean temperature over a slice by using the `BTESaveCC` command (see BTESaveCC on page 56):

```
BTESaveCC tdrFile=Temperature1D_out.tdr models=LatticeTemperature
```

Solving at One Bias

The solution of the subband-BTE at a specific bias is initiated using the `Solve` command. Similar to calculations using Sentaurus Band Structure, the bias on each contact is set using the `V(<contact>)` syntax, where `<contact>` refers to the contact name. For example, to initiate a solve at specific biases on the source, drain, and gate contacts, use the following command:

```
Solve V(gate)=1.0 V(drain)=1.0 V(source)=0.0
```

If the bias on a particular contact is not specified explicitly in the `Solve` command, then the previously specified bias is used. By default, the bias on all contacts is set to 0 at the start of the simulation.

Initial Guess

If there are large steps in the applied bias, the solution of the subband-BTE in the presence of sophisticated scattering models might take many iterations to reach a convergent result. This is computationally inefficient, since the major correction is in the electrostatics and can be obtained more efficiently by starting from an initial guess, which provides a better starting reference in terms of the electrostatic solution for the solver. For example, to provide an initial guess using the ballistic Subband-BTE solver, specify the following commands:

```
## Remove all scattering mechanisms
Physics material=all ScatteringModel removeAll

## Define solver and solve ballistic subband-BTE
Physics slicedChannel=channel1 eBTESolver=Numeric
Solve V(drain)=1.0 V(gate)=1.0

## Define scattering mechanism
...

## Redefine solver and solve subband-BTE in the presence of scattering
Physics slicedChannel=channel1 eBTESolver=Numeric
Solve V(drain)=1.0 V(gate)=1.0
```

Convergence Criteria

During the solution of the subband-BTE, the Schrödinger equation, the subband-BTE, and the Poisson equation are solved in an iterative fashion. The overall iteration of these equations is divided into an outer iteration of the Poisson equation and an inner iteration of the subband-BTE, after a single solve of the Schrödinger equation. This is shown in Figure 1 on page 12.

Convergence criteria for the outer Poisson equation are set by the following arguments of the `Math` command:

- The `currentConvTol` argument determines the relative change in terminal currents between subsequent Poisson iterations below which the subband-BTE system is considered to have converged.
- The `potentialUpdateTolerance` argument, set in units of $k_B T/q$, determines the maximum-allowed change in the potential below which the Poisson equation is considered to have converged. For the subband-BTE system of equations, it is recommended to set this argument to a value of approximately 3.0e-3.

The subband-BTE system is considered to have converged when one of these convergence criteria is met.

For example, if `potentialUpdateTolerance=3.0e-3` and `currentConvTol=1.0e-3`, then a bias point for the subband-BTE is considered converged if the maximum change in the potential at a particular Poisson Newton iteration is below 3.0e-3 × ($k_B T/q$) or the change in terminal current relative to the previous Newton iteration is below 1.0e-3.

By default, `currentConvTol=0.0`, meaning that convergence is completely controlled by the `potentialUpdateTolerance` argument.

Convergence criteria for the inner solution of the subband-BTE are set using the following arguments:

- `distFuncUpdateTolerance` for the distribution functions
- `rtaUpdateTolerance` for the local Fermi level used in the RTA scattering model

Note:

It is recommended to use the default values for these arguments.

The solution of the subband-BTE requires an initial guess for the distribution functions. For the first several Poisson iterations, this guess is supplied by solving the subband-BTE using only the RTA scattering model. After these initial Poisson iterations, the initial guess is supplied by the solution of the subband-BTE from the previous Poisson iteration. The exact number of Poisson iterations for which the RTA solution is used as an initial guess is set by the `iterationsForRTAGuess` argument of the `Math` command. It is recommended to use its default value. For details about these arguments, see Math on page 43.

Output

The primary output of the solution of the Subband-BTE solver is the current, in ampere, flowing through the left and right contacts at the end of the structure. These current values are printed to the screen at the end of a `Solve` command, and the values also are stored in the bias log file. Furthermore, in the bias log file, when a nonzero lumped resistance is

specified, in addition to the applied bias that is stored under `V(<contact>)`, the so-called inner voltage on the inner end of the lumped resistor is stored under `Vinner(<contact>)`.

In addition to the current, you can save several different TDR files with data from the solution of the Subband-BTE solver.

2D TDR File With Fields Over Channel Coordinate–Energy Space

You can use the `BTESaveE` command to save a 2D TDR file with fields that are defined on the channel coordinate–energy grid. These fields include quantities such as the distribution function, the current spectrum, and the density-of-states (DOS) in each subband, or in each valley, or in total (see BTESaveE on page 53).

2D TDR File With Fields Over Channel Coordinate–k-Space

You can use the `BTESaveK` command to save a 2D TDR file with fields that are defined over channel coordinate–*k*-space (see BTESaveK on page 55).

TDR (XY) File With Fields Versus Channel Coordinate

You can use the `BTESaveCC` command to save a TDR (xy) file with fields versus the channel coordinate. The fields that can be saved include the total inversion charge, the total current, the total carrier velocity, as well as these quantities per subband. In addition, the occupancy of each subband can be saved (see BTESaveCC on page 56).

Slice-Related Fields

Each slice that is used in the solution of the subband-BTE is basically a 2D cross section. Similar to the direct treatment of a 2D device structure using Sentaurus Band Structure, various fields across this 2D cross section, as well as fields in 1D *k*-space, can be saved using two commands.

2D TDR File With Fields Over XY Real Space

You can use the `SaveSlice` command to save quantities from the Schrödinger solver on a slice such as the subband energy and the wavefunctions, as well as quantities such as the conduction band energy (see SaveSlice on page 58).

TDR (XY) File With Fields Over 1D k-Space

You can use the `SaveSliceK` command to save the 1D *k*-space dispersion for each subband computed by the Schrödinger solver (see SaveSliceK on page 59).

References

[1] S. Jin, T.-W. Tang, and M. V. Fischetti, “Simulation of Silicon Nanowire Transistors Using Boltzmann Transport Equation Under Relaxation Time Approximation,” *IEEE Transactions on Electron Devices*, vol. 55, no. 3, pp. 727–736, 2008.

[2] D. Esseni, P. Palestri, and L. Selmi, *Nanoscale MOS Transistors: Semi-Classical Transport and Applications*, Cambridge: Cambridge University Press, 2011.

2

Example: Getting Started

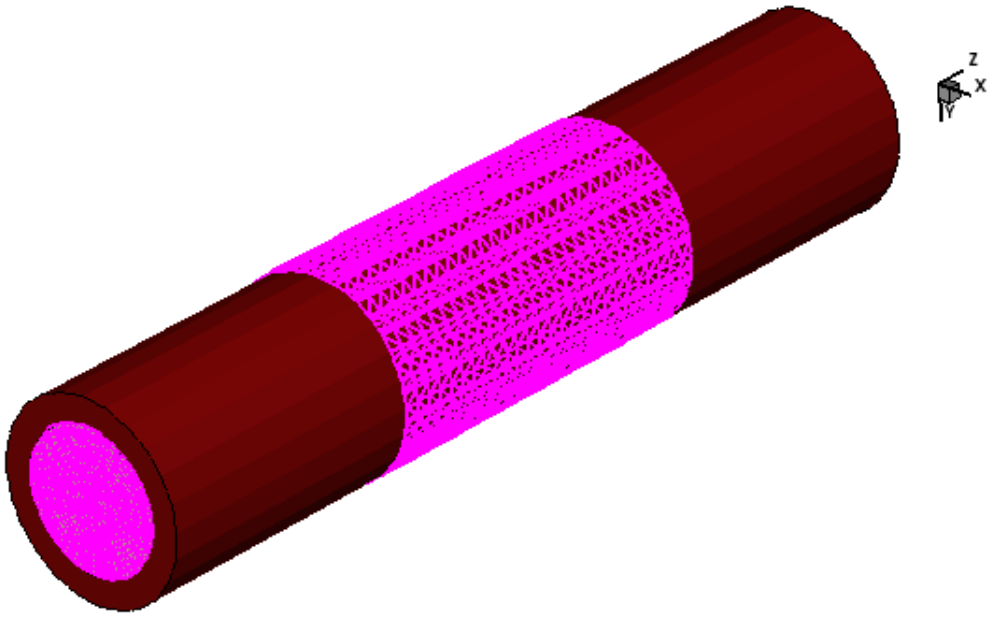
This chapter presents an example that uses the Subband-BTE solver to compute an I_d–V_g curve for a silicon NMOS nanowire.

This chapter describes the command file for simulating ballistic transport in a silicon nanowire device using the Subband-BTE solver, including sections of the command file that load the device structure, set up the models, specify the sliced channel, and perform the I_d–V_g simulation.

Device Structure

Figure 3 shows the device structure for this example.

Figure 3 Silicon nanowire device for the example



The device is a cylindrical silicon NMOS nanowire with a diameter of 5 nm, a gate length of 13 nm, and source and drain extension regions of length 10 nm. The source and drain are uniformly doped n-type to 2e20 cm^{-3}. This example computes an I_d–V_g curve at V_d = 0.6 V in the ballistic limit using the `Numeric` Subband-BTE solver.

Loading the Device Structure

```
LoadDevice tdrFile=nanowire3D_0.tdr
```

The `LoadDevice` command loads the device structure from the `nanowire3D_0.tdr` file. After loading, the tool automatically sets up the default models and parameters in the silicon and oxide regions.

Setting Up the Model and the Classical Solve

```
# Simple model for hole density
Physics material=Silicon hBulkDensity=hFermiDensity Nv=3.10e19

# Remove all scattering models for ballistic simulation
Physics material=all ScatteringModel removeAll

# Use Neumann boundary condition on source and drain
Physics contact=source type=Neumann
Physics contact=drain  type=Neumann

# Set workfunction
Physics contact=gate workfunction=4.25

# Set orientation for <110> channel
Physics xDirection=[list 1 1 0] yDirection=[list 0 0 1]

# Classical solve
Solve
```

These commands set up particular models and parameters, starting with specifying a simple model for the hole density. Then, all the scattering models are removed, so that a ballistic simulation can be performed. The boundary condition type for the source and drain contacts is set to Neumann. The gate workfunction is set to 4.25 V.

After the required models are specified, a classical solve of only the Poisson equation is performed in equilibrium, that is, zero bias on all contacts. A *classical solve* means that the Schrödinger equation is not used. The resulting classical solution serves as the initial guess for the first solve using the Schrödinger equation.

Specifying the Sliced Channel

```
# Create sliced channel over all device for si1 region
Math slicedChannel name=channel1 regions=[list si1]

# Set up parabolic Schrodinger on the sliced channel
Physics slicedChannel=channel1 eSchrodinger=Parabolic \
   valleys=[list Delta1 Delta2 Delta3] Nsubbands=8 Nk=16 Kmax=0.6 \
   a0=5.43e-8 correction=3

# Select Numeric Subband-BTE solver and set energy grid parameters
Physics slicedChannel=channel1 eBTESolver=Numeric deltaE=4.0e-3 \
   resolveTOBEnergy=1 minDeltaE=1.0e-4
```

This section of the command file specifies the sliced channel, the Schrödinger solver, and the Subband-BTE solver.

The `Math slicedChannel` command creates a sliced channel named `channel1` over the region named `si1` throughout the entire device.

The first `Physics slicedChannel` command specifies that a parabolic Schrödinger solver must be used on all slices of the sliced channel. This command refers to the `Delta1`, `Delta2`, and `Delta3` valley models that were created by default during the `LoadDevice` command.

The second `Physics slicedChannel` command changes the Subband-BTE solver to `Numeric` and specifies some energy grid parameters.

Performing the I_d–V_g Simulation

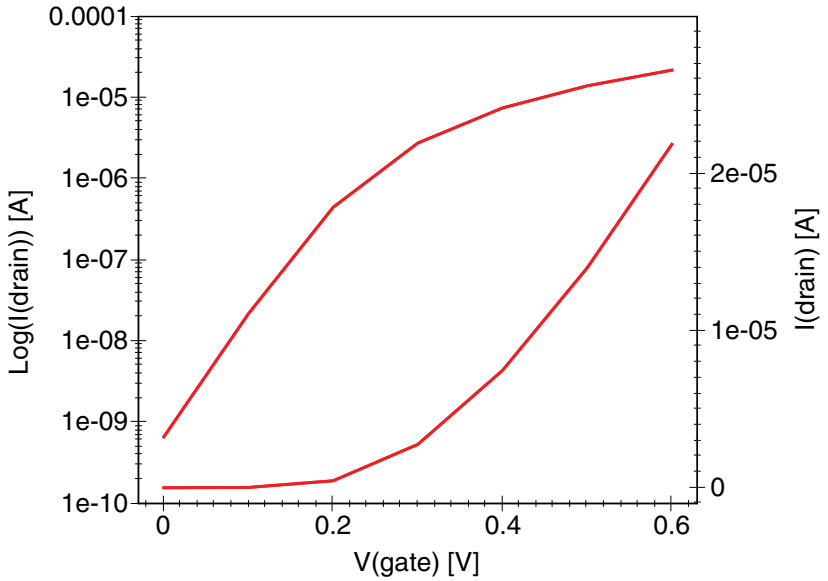
```
# Set some convergence parameters. It continues to next bias if not
# converged
Math potentialUpdateTolerance=6.0e-3 iterations=30 doOnFailure=0

# Solve at equilibrium and then at required drain bias
Solve V(gate)=0.0 V(drain)=0.0

Solve V(drain)=0.6

# Ramp gate using a Tcl foreach loop
foreach Vg [list 0.0 0.1 0.2 0.3 0.4 0.5 0.6] {
   Solve V(gate)=$Vg
   AddToLogFile name=Id value=[GetLast name=I(drain)]
}
```

This section of the command file computes the I_d–V_g curve. The `Math` command specifies the convergence tolerance for the Poisson equation. The number of allowed iterations for

each bias is set to 30, and the `doOnFailure` argument is set such that the tool will continue to the next bias, even if convergence criteria are not met.

The series of `Solve` commands write data to the default bias log file named `subbandBTE_Quickstart.plt`. An extra user-defined quantity named `Id` is added to the bias log file to plot the I_d–V_g curve more easily in Sentaurus Visual. Figure 4 shows the resulting I_d–V_g curve.

Figure 4 Ballistic I_d–V_g curve from the example



Calculating Scattering-Limited Transport Parameters

From the solution of the subband-BTE, scattering-limited mobilities and diffusivities can be calculated by using the moments of the inverse scattering operator (MISO) method [1][2]. While other methods for scattering-limited mobility calculation such as Kubo–Greenwood rely on linear response theory and yield the low-field mobility as the result, the MISO method is the generalization to the non-equilibrium case and can be used to study high-field effects.

Mobility and diffusivity are computed using the first moment of the inverse scattering operator, that is, the projection of the inverse scattering operator onto the velocity.

Based on Equations (3.10) and (3.11) in [1], the expressions for diffusivity and mobility can be written, respectively, as:

$$D(z) = -\frac{1}{N_{\text{tot}}(z)}\sum_{n}\frac{1}{2\pi}\int dk\sum_{n'}\frac{1}{2\pi}\int dk'\nu_{n}(z,k)S^{-1}_{n,n'}(z,k,k')\nu_{n'}(z,k')f_{n'}(z,k')$$

$$\mu(z) = \frac{q}{N_{\text{tot}}(z)}\sum_{n}\frac{1}{2\pi}\int dk\sum_{n'}\frac{1}{2\pi}\int dk'\nu_{n}(z,k)S^{-1}_{n,n'}(z,k,k')\nu_{n'}(z,k')\frac{\partial f_{n'}(z,k')}{\partial \varepsilon_{n'}(z,k')} \quad (11)$$

Note:

The MISO method yields the scattering-limited diffusivity and mobility, but not the ballistic mobility.

This section of the input file specifies the command for calculating the scattering-limited transport parameters, which can be performed as a postprocessing step after the subband-BTE has been solved using the `Solve` command. It requires a unique name for the scattering models and the previously defined `SlicedChannel` object.

If you need to define multiple scattering models, you can make arbitrary sub-selections of scattering models using the `scatteringModelNames` input parameter of the command `ComputeTransportParametersFromBTESolver`.

Example

```
# Define scattering model
...

# Define sliced channel and BTE solver, see previous sections
...

  # Solve the subband BTE
Solve

# Calculate the scattering-limited transport parameters, assuming the
# following scattering models have been previously defined:
# ElectronSR: name of surface roughness scattering model
# AcousticPH: name of acoustic intravalley phonon scattering model
# ivg1_EMS:   name of intervalley g-type phonon scattering model
#             for phonon emission
# ivg1_ABS:   name of intervalley g-type phonon scattering model
#             for phonon emission
ComputeTransportParametersFromBTESolver slicedChannel=channel1 \
   scatteringModelNames=[list [list ElectronSR] \
   [list AcousticPH ivg1_EMS ivg1_ABS] ] outputNames=[list SR PH]

# Save output
BTESaveCC tdrFile=filename.tdr models=[list Mobility Diffusivity]
```

In addition to the quantities you specified, the total scattering-limited transport parameters are always computed when you specify the `Mobility` or `Diffusivity` keyword in the `BTESaveCC` command (see BTESaveCC on page 56).

The total scattering-limited transport parameters are computed with respect to all specified scattering mechanisms, in contrast to the user-defined sub-selection of scattering models, which can be requested by the `ComputeTransportParametersFromBTESolver` command. You will obtain the following data fields in the TDR file:

- `MobilitySR`, `MobilityPH`, and `MobilityTotal`
- `DiffusivitySR`, `DiffusivityPH`, and `DiffusivityTotal`

See ComputeTransportParametersFromBTESolver on page 51.

Saving TDR Files

```
# Save 3D TDR file
Save tdrFile=subbandBTE_Quickstart.tdr \
   models=[list DopingConcentration eDensity hDensity  \
   ConductionBandEnergy eQuasiFermiEnergy]

# Save fields over ChannelCoord-Energy
BTESaveE tdrFile=subbandBTE_Quickstart_CC_E.tdr \
   models=[list Delta1_0_DistributionFunction  \
      Delta1_0_CurrentSpectrum Delta1_0_SubbandEnergy Delta1_0_DOS]

# Save distribution function over ChannelCoord-k
BTESaveK tdrFile=subbandBTE_Quickstart_CC_K.tdr \
   models=[list Delta1_0_DistributionFunction]

# Save subband quantities over ChannelCoord
BTESaveCC tdrFile=subbandBTE_Quickstart_CC.tdr \
   models=[list NinvTotal CurrentTotal VelocityTotal \
      Delta1_0_Occupancy Delta3_0_Occupancy]

# Save fields over a slice
SaveSlice tdrFile=subbandBTE_Quickstart_Slice.tdr =20.0e-3 \
   channelCoord models=[list eDensity Delta1_0_Wavefunction]

# Save dispersion at a slice
SaveSliceK tdrFile=subbandBTE_Quickstart_SliceK.tdr  \
   channelCoord=20.0e-3 models=[list Delta1_0_Dispersion]
```

This section of the command file writes various TDR files with different types of data from the solution of the subband-BTE:

- The `Save` command writes a 3D TDR file with the 3D device structure and the selected models.
- The `BTESaveE` command writes a 2D TDR file over channel coordinate–energy space. In this example. the distribution function, the current spectrum, the subband energy, and DOS for the `Delta1_0` subband are included. Figure 5 shows the plot of the resulting distribution function.
- The `BTESaveK` command saves a TDR file over channel coordinate–*k*-space with only the distribution function for the `Delta1_0` subband.
- The `BTESaveCC` command saves a TDR (xy) file with the selected models versus the channel coordinate. Figure 6 shows the plot of the resulting total inversion charge (NinvTotal) and the carrier velocity (VelocityTotal) along the channel.
- The final commands, `SaveSlice` and `SaveSliceK`, save real-space and *k*-space quantities, respectively, from the slice located closest to the channel coordinate of 20 nm.

Figure 5 Plot of the distribution function in the Delta1_0 subband from the BTESaveE command

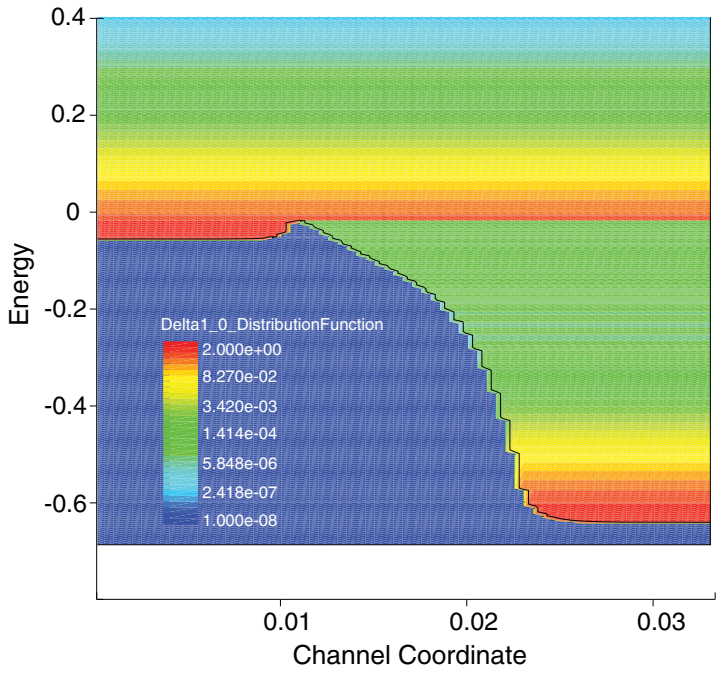
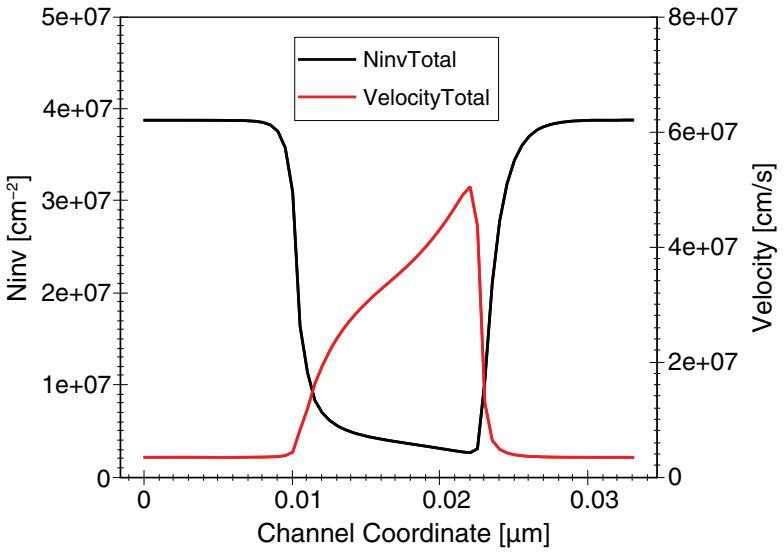


Figure 6 *Plot of the total inversion charge and carrier velocity along the nanowire channel from the BTESaveCC command*



References

[1] S. C. Brugger, *Computation of Semiconductor Properties Using Moments of the Inverse Scattering Operator of the Boltzmann Equation*, PhD thesis, ETH, Zurich, Switzerland, 2006.

[2] S. C. Brugger, A. Schenk, and W. Fichtner, “Moments of the Inverse Scattering Operator of the Boltzmann Equation: Theory and Applications,” *SIAM Journal on Applied Mathematics*, vol. 66, no. 4, pp. 1209–1226, 2006.

3

Example: Coupling Sentaurus Device and Subband-BTE Solver

This chapter presents an example of a coupled Sentaurus Device–Subband-BTE solver simulation to compute the I_d–V_g curve of a non-extruded nanowire FET.

This chapter describes the changes to a standalone Sentaurus Device QTX command file required for a coupled Sentaurus Device–Subband-BTE solver simulation. Most of the syntax is identical, so only those sections that differ are described here.

See *Sentaurus™ Device User Guide*, External Boltzmann Solver, for the required modifications to the Sentaurus Device command file.

Device Structure

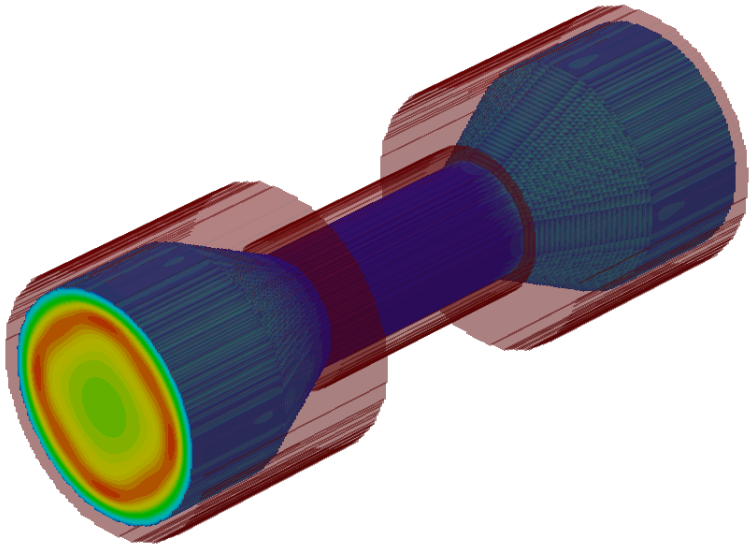
Figure 7 shows the device structure for this example. It is a cylindrical non-extruded n-type nanowire FET with a diameter of 7 nm in the source and drain extension regions, and a diameter of 5nm in the channel. The diameter changes linearly over a length of 5 nm. The source and drain extensions are uniformly doped at 1e20 nm^{-3}. This example computes an I_d–V_g curve at V_d = 0.6 V in the ballistic limit.

In contrast to the mesh of the structure in Chapter 2 on page 31, the mesh here can be an unstructured Delaunay mesh as long as it fulfills one additional constraint: it must contain several well-defined planes of mesh points perpendicular to the z-axis, which still describes the main transport axis.

These planes can be inserted with the `line` command when using Sentaurus Process or the `zCuts` command when creating the mesh using Sentaurus Mesh. Refer to the respective product documentation for details.

In the example, 31 planes along the z-axis are introduced equidistantly every 1 nm. The source, drain, and channel are 10 nm long each, so the length of the total device is 30 nm. The gate contact is 10 nm long and is wrapped all around the channel region. The SiO_2 thickness is 0.8 nm.

Figure 7 Example of a nanowire FET with a non-extruded device structure



Specifying the Sliced Channel

As this example demonstrates a non-extruded structure, automated slice extraction is not possible. Here, the slice locations must be specified at the time of slice creation. For details, see Creating a Sliced Channel on page 15.

For the example here, use the following commands:

```
# Specify coordinate list of the mesh planes created during structure
# generation used to extract slices from the 3D device (unit: um)

set sliceCoordList [list 0.0 0.001 0.002 ... 0.029 0.03]

Math slicedChannel name=channel1 regions=[list si1] \
                   sliceCoords=$sliceCoordList
```

The setup of the valley models, the `eSchroedinger` solver, and the `eBTESolver` are identical to that in Specifying the Sliced Channel on page 33.

Performing the I_d–V_g Simulation

To run the I_d–V_g simulation as a coupled Sentaurus Device–Subband-BTE solver simulation, the `Solve` command in the Sentaurus Device QTX command file is replaced with the following command:

```
ConnectBTEToMasterProcess connectionName=n@node@ slicedChannel=channel1
```

This command establishes the connection to Sentaurus Device. The connection name must be a unique string and must also be specified in the Sentaurus Device command file, to establish the connection successfully.

Note:

Sentaurus Device is completely responsible for controlling the voltages.

After the successful voltage ramp and Sentaurus Device has finished its calculation, it returns control to Sentaurus Device QTX, and the usual execution of Tcl commands resumes.

Saving TDR Files

To print plot (`.plt`) and TDR (`.tdr`) files with the results of the `BTESolver`, the command `ConnectBTEToMasterProcess` has an additional option, where a Tcl function can be specified that handles the Subband-BTE solver output. Whenever a bias point converged, Sentaurus Device notifies Sentaurus Device QTX, and this function is called and the data is saved.

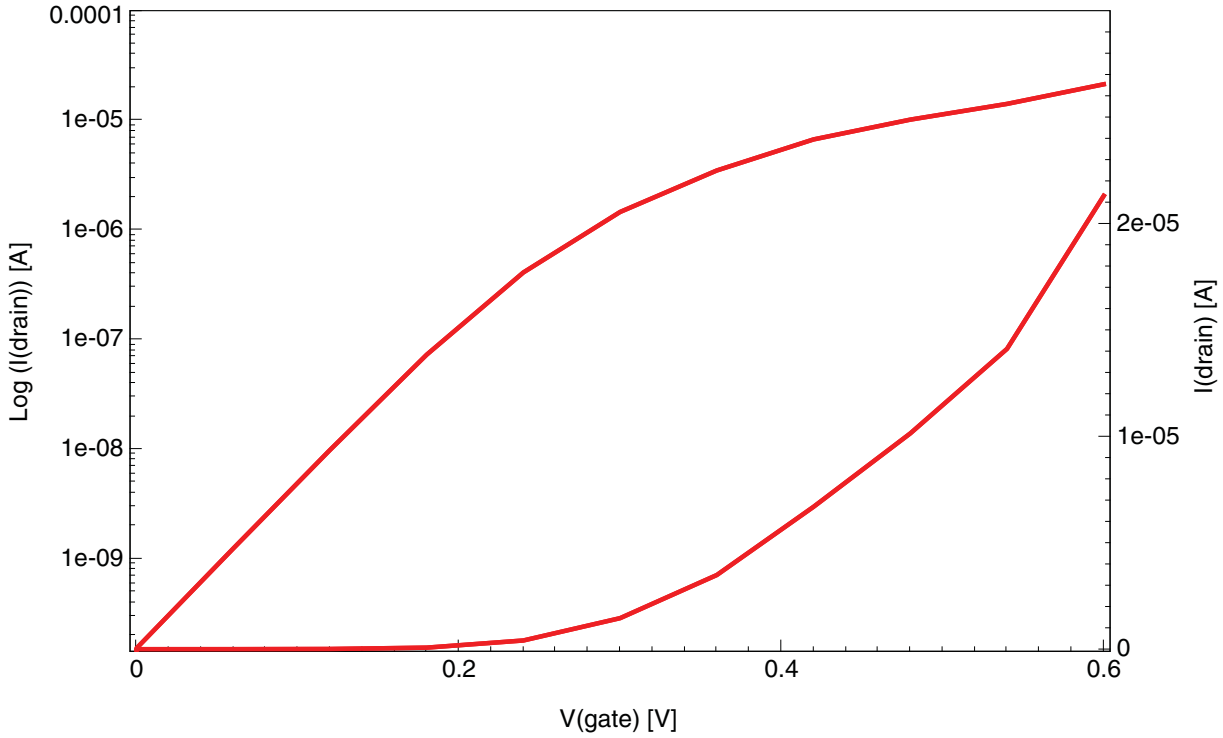
For example:

```
proc output {file_prefix} {
   upvar valleys valleys
   AddToLogFile name=Id value=[GetLast name=I(drain)]
   Save tdrFile=${file_prefix}_bte.tdr \
      models=[list eDensity DopingConcentration]

   set models {}
   foreach valley $valleys {
      for {set subband 0} {$subband < @Nsubbands@} {incr subband} {
         lappend models ${valley}_${subband}_Ninv
         lappend models ${valley}_${subband}_SubbandEnergy
      }
   }

   BTESaveCC tdrFile=${file_prefix}_bte_CC.tdr models=$models
}

ConnectBTEToMasterProcess connectionName=n@node@ \
   slicedChannel=channel1 outputFunctionName=output
```

Note:

The Tcl function output must be defined before specifying the `ConnectBTEToMasterProcess` command. Otherwise, it is not known in that scope. All the `Save` commands of a regular Subband-BTE solver simulation are available. The only difference is that, in this context, they are called using a Tcl function because Sentaurus Device controls the voltages.

Figure 8 Ballistic I_d–V_g curve from the example



4

Commands of the Subband-BTE Solver

This chapter summarizes the commands of the Subband-BTE solver.

Math and Physics Commands

This section describes the commands used to specify the geometry and the physics of a sliced channel as well as the convergence criteria for the solution of the subband-BTE.

Math

This command sets the convergence criteria for solving the subband-BTE and also activates the self-heating capability.

The basic `Math` command sets the convergence criteria for solving the subband-BTE using the `Numeric` Subband-BTE solver along with the RTA scattering model. The subband-BTE is considered converged when both criteria specified by `distFuncUpdateTolerance` and `rtaUpdateTolerance` are met.

The `iterationsForRTAGuess` argument controls the initial guess for the solution of the subband-BTE. Setting this to a large value can improve convergence at the beginning of a `Solve`; however, it might result in a large overall number of Poisson iterations.

Syntax

```
Math [currentConvTol=Double] [distFuncUpdateTolerance=Double] \
   [iterationsForRTAGuess=Integer] \
   [potentialThresholdForSimplifiedEV=Double] \
   [potentialUpdateTolerance=Double] \
   [rtaUpdateTolerance=Double] [selfHeating=Boolean] \
   [sliceCoords=List] [subbandBTELinearSolver=Boolean]
```

Argument	Description	Default	Unit
`currentConvTol`	Sets the tolerance on the relative error in the terminal current compared to the previous Newton iteration.	0.0	1
`distFuncUpdateTolerance`	Sets the tolerance on the infinity norm for the update to the distribution functions.	1.0e-8	1
`iterationsForRTAGuess`	Sets the number of iterations at the start of a solve for which the solution of the subband-BTE with only relaxation time approximation (RTA) scattering is used as an initial guess.	5	1
`potentialThresholdFor SimplifiedEV`	Sets the threshold for activating a simplified eigensolve (see Specifying a Simplified Eigensolve on page 24).	0.0	V
`potentialUpdate Tolerance`	Sets the convergence tolerance applied to the potential update in units of the thermal voltage.	1.0e-5	$k_{\mathrm{B}}T/q$
`rtaUpdateTolerance`	Sets the tolerance on the infinity norm for the update to the local Fermi levels for the RTA scattering model.	1.0e-2	eV
`selfHeating`	Specifies whether to activate (`1`) or to not activate (`0`) self-heating.	0	–
`sliceCoords`	Specifies a list of z-coordinates, where slices should be extracted. This argument is required when dealing with non-extruded structures in coupled simulations with Sentaurus Device.	–	µm
`subbandBTELinearSolver`	Selects the iterative linear solver for the subband-BTE system. Options are: • `0`: Use ILS. • `1`: Use bitlis. The iterative linear solver *bitlis* is ideal for large systems with many subbands, as it typically reduces the memory and solve time significantly compared to ILS for such systems.	1	–

Examples

Set the update tolerance on the distribution functions to 1.0e-7 and on the RTA Fermi levels to 1.0e-3 eV:

```
Math distFuncUpdateTolerance=1.0e-7 rtaUpdateTolerance=1.0e-3
```

Math slicedChannel

This command creates a sliced channel within a 3D device.

The `Math slicedChannel` command creates a sliced channel within the 3D device that has been already loaded using the `LoadDevice` command. This command automatically creates the 2D slices along the channel and a nonlocal area for each slice. The name of the nonlocal area is set automatically to `<slicedChannel>.NL`. For example, if the sliced channel is named `channel1`, the nonlocal is named `channel.NL`. As in the `Math nonlocal` command, only mesh elements contained within the specified geometry are included.

Syntax

```
Math slicedChannel name=String \
   [channelAxis=String] [minX=Double] [maxX=Double] \
   [minY=Double] [maxY=Double] [minZ=Double] [maxZ=Double] \
   [regions=List]
```

Argument	Description	Default	Unit
`slicedChannel`	Indicates the creation of a new sliced channel.	–	–
`name`	Sets the name of the sliced channel.	–	–
`channelAxis`	Sets the device axis along the channel. The axis must be `z`.	`z`	–
`minX`	Specifies the minimum x-coordinate of the sliced channel.	$-\infty$	µm
`maxX`	Specifies the maximum x-coordinate of the sliced channel.	∞	µm
`minY`	Specifies the minimum y-coordinate of the sliced channel.	$-\infty$	µm
`maxY`	Specifies the maximum y-coordinate of the sliced channel.	∞	µm
`minZ`	Specifies the minimum z-coordinate of the sliced channel.	$-\infty$	µm
`maxZ`	Specifies the maximum z-coordinate of the sliced channel.	∞	µm
`regions`	Specifies a list of regions to be included in the sliced channel.	All regions	–

Examples

Create a new sliced channel named `channel1` (slices along the channel axis, which defaults to `z`, are extracted automatically within the `si1` and `ox1` regions):

```
Math slicedChannel name=channel1 regions=[list si1 ox1]
```

Physics contact

This command sets the properties of the specified contact.

The `Physics contact` command sets the properties of the specified contact. For the Subband-BTE solver, the `type` argument can be used to set the required Neumann boundary condition for the solution of the subband-BTE on the left and right contacts of each sliced channel.

You can use the `resist` argument to specify a lumped resistance for a contact through which current flows.

Syntax

```
Physics contact=String [resist=Double] [schottkyBarrier=Double] \
   [type=String] [workfunction=Double]
```

Argument	Description	Default	Unit
`contact`	Sets the name of the contact.	–	–
`resist`	Sets the lumped resistance.	0.0	Ω
`schottkyBarrier`	Sets the Schottky barrier value on the contact–metal semiconductor interface.	–	eV
`type`	Selects the type of boundary condition used for the Poisson equation at the contact. Specify `Neumann` to set Neumann boundary condition.	`Dirichlet`	–
`workfunction`	Sets the metal workfunction value for the contact.	4.1	eV

Examples

Set the boundary condition type on the source and drain contacts to Neumann:

```
Physics contact=source type=Neumann
Physics contact=drain  type=Neumann
```

Physics slicedChannel

This command specifies the Schrödinger solver or the Subband-BTE solver to use on a sliced channel.

The `Physics slicedChannel` command is used to specify the Schrödinger solver or the Subband-BTE solver to use on a sliced channel, depending on which parameters are used.

To specify the Schrödinger solver to use on all slices in a sliced channel, use the `eSchrodinger` or `hSchrodinger` argument. The use of this argument is identical to setting up a Schrödinger solver using the `Physics nonlocal` command, and all of the Schrödinger-related arguments can be used here as well.

To specify the Subband-BTE solver to use on a sliced channel, use the `eBTESolver` or `hBTESolver` argument.

Syntax

To specify the Schrödinger solver to use on all slices in a sliced channel, use:

```
Physics slicedChannel=String eSchrodinger=String ...
```

To specify the Subband-BTE solver to use on a sliced channel, use:

```
Physics slicedChannel=String (eBTESolver=String | hBTESolver=String) \
   [approximateJacobian=Boolean] \
   [combineSpinOfDegenerateStates=Boolean]\
   [deltaE=Double] [interBranchCoupling=Boolean] [minDeltaE=Double] \
   [NkForScreening=Integer] [reorderDispersion=Boolean] \
   [resolveTOBEnergy=Boolean] \
   [simplifiedEVDegenTol=Double] \
   [subbandScatteringRange=Integer] \
   [tau=Double] [tdfDegenTol=Double] \
   [useKdependentWF=Boolean] \
   [useKdependentWFForBTEDensity=Boolean] \
   [useOverlapInterpolation=Boolean] \
   [useParabolicFitForToB=Boolean] \
   [whenToComputeScreening=String]
```

Argument	Description	Default	Unit
`slicedChannel`	Sets the name of the sliced channel.	–	–
`eBTESolver`	Selects the type of Subband-BTE solver to use for solving the subband-BTE on the sliced channel. Options are `AnalyticBallistic` and `Numeric`.	`Numeric`	–

Argument	Description	Default	Unit
`hBTESolver`	Selects the type of Subband-BTE solver to use for solving the subband-BTE on the sliced channel. Options are `AnalyticBallistic` and `Numeric`.	`Numeric`	–
`eSchrodinger`	Selects the name of the Schrödinger solver to use on all slices in the sliced channel. For the Subband-BTE solver, this must be `Parabolic` or `2kp`.	–	–
`hSchrodinger`	Selects the name of the Schrödinger solver to use on all slices in the sliced channel. For the Subband-BTE solver, options are `6kp`, `3kp`, and `Parabolic`.	–	–
`approximateJacobian`	Specifies whether an approximate Jacobian is used to solve the BTE, thereby decreasing memory consumption.	1	–
`combineSpinOfDegenerateStates`	Specifies whether to reduce the cost of the Boltzmann transport problem for valley models with explicit spin representation. Options are: • `0`: Do not use the degeneracy of the subband dispersion. • `1`: Use the degeneracy of the subband dispersion.	0	–
`deltaE`	Sets the nominal energy grid spacing.	4.0e-3	eV
`interBranchCoupling`	Specifies the correct treatment of classical phase-space reversal points and heterostructure-like transitions between dispersions on adjacent slices.	1	–
`minDeltaE`	When the top of the barrier for each subband is refined, this argument specifies the minimum energy spacing to use.	1.0e-4	eV
`NkForScreening`	Sets the number of k-points to use when evaluating screening.	32	–

Argument	Description	Default	Unit
`reorderDispersion`	The default order of the subbands for each *k*-point is based on the energy value. If reordering of the subband dispersion is switched on, then the order of the subbands is determined through wavefunction overlap in *k*-space. Options are: • `0`: Do not reorder subband dispersion. • `1`: Reorder subband dispersion.	0	–
`resolveTOBEnergy`	Specifies whether the energy grid near the top of each subband barrier is refined. Options are: • `0`: Do not refine. • `1`: Refine.	1	–
`simplifiedEVDegenTol`	Sets the tolerance used to capture degenerate eigenvalues when using a simplified eigensolve (see Specifying a Simplified Eigensolve on page 24).	1.e-4	eV
`subbandScatteringRange`	Specifies the allowed subband scattering range between the initial and final subbands. A value less than 0 indicates that scattering to all valid subbands is allowed. This is the case by default.	–1	–
`tau`	Sets the relaxation time for the built-in RTA scattering model.	1.0e-10	s
`tdfDegenTol`	When you specify a positive value, this value sets the subband energy difference tolerance for screening inter-subband transitions with the tensor Lindhard screening model. When you specify a negative value, a special algorithm is used to automatically select the inter-subband transitions that are treated.	–1	eV
`useKdependentWF`	This argument is forwarded to the Schrödinger solver on each slice and specifies that the wavefunctions should be computed at each *k*-point in the *k*-space grid. Options are: • `0`: Only evaluate wavefunctions at the subband minimum. • `1`: Evaluate wavefunctions at each *k*-point.	0	–

Argument	Description	Default	Unit
`useKdependentWFForBTEDensity`	Specifies whether *k*-dependent wavefunctions are used to compute the carrier density.	0	–
`useOverlapInterpolation`	Specifies whether to use linear interpolation between the *k*-dependent isotropic form factors in the scattering models. Options are: • `0`: Do not use linear interpolation. • `1`: Use linear interpolation.	0	–
`useParabolicFitForToB`	Specifies whether a special algorithm is used to locate more accurately the top of the source–drain barrier.	0	–
`whenToComputeScreening`	Controls when the selected screening model is computed during a bias solve. Options are: • `EveryIteration` computes for every Poisson Newton iteration. • `FirstIteration` computes only during the first Poisson Newton iteration. See Runtime Speedup With Screening on page 20.	`Every Iteration`	–

Examples

Specify that a parabolic Schrödinger solver must be used on all slices of the sliced channel named `channel1` (the usual set of arguments for setting up a Schrödinger solver is specified as well):

```
Physics slicedChannel=channel1 eSchrodinger=Parabolic \
   valleys=[list Delta1 Delta2 Delta3] Nsubbands=8 Nk=16 Kmax=0.6 \
   a0=5.43e-8
```

Change the Subband-BTE solver for `channel1` to `AnalyticBallistic`, and change the nominal energy grid spacing to 5 meV:

```
Physics slicedChannel=channel1 eBTESolver=AnalyticBallistic \
   deltaE=5.0e-3
```

Physics thermalContact

This command sets the properties of the specified thermal contact. The type of the thermal contact can be either `Dirichlet` or `Neumann`. If the type is `Dirichlet`, then the corresponding temperature can also be set using `contactTemperature`.

Syntax

```
Physics thermalContact=String [contactTemperature=Double] \
   [thermalResist=Double] [type=String]
```

Argument	Description	Default	Unit
`thermalContact`	Sets the type of thermal contact.	–	–
`contactTemperature`	Sets the temperature of the contact.	300	K
`thermalResist`	Sets the value of the thermal resistance connected to the contact.	0.0	K/W
`type`	Specifies the boundary condition of the thermal contact. Options are: • `Dirichlet` • `Neumann`	`Dirichlet`	–

Examples

```
Physics thermalContact=source contactTemperature=300
Physics thermalContact=drain  contactTemperature=300
Physics thermalContact=gate   type=Neumann
```

Computing Transport Parameters

This section describes the command to be used to calculate scattering-limited transport parameters (see Calculating Scattering-Limited Transport Parameters on page 34).

ComputeTransportParametersFromBTESolver

This command computes scattering-limited mobilities and diffusivities for arbitrary sub-selections of scattering models in a postprocessing step to the `Solve` command.

Syntax

```
ComputeTransportParametersFromBTESolver outputNames=List \
   scatteringModelNames=List slicedChannel=String
```

Argument	Description	Default	Unit
`outputNames`	Specifies a list of the names of each sub-selection, used to label the output quantities.	–	–
`scatteringModelNames`	Specifies a list of the sub-selections of scattering models.	–	–
`slicedChannel`	Sets the name of the sliced channel.	–	–

Establishing a Connection to Sentaurus Device

This section describes the command that must be used to establish a connection to Sentaurus Device when performing a coupled Sentaurus Device–Subband-BTE solver simulation.

ConnectBTEToMasterProcess

This command basically replaces the `Solve` command.

Syntax

```
ConnectBTEToMasterProcess connectionName=String \
   outputFunctionName=String slicedChannel=String
```

Argument	Description	Default	Unit
`connectionName`	Specifies a unique string that must be used in both the Sentaurus Device QTX and Sentaurus Device command files.	–	–
`outputFunctionName`	Sets the name of the Tcl function that handles the syntax for saving TDR (`.tdr`) and `.plt` files.	–	–
`slicedChannel`	Sets the name of the sliced channel.	–	–

Saving TDR Files

Several types of data and TDR files can be saved. Each type of TDR file is saved using a command specific to the type of data as described here.

Specifying Subbands and Subband IDs

Many quantities are defined on a subband basis. Therefore, you must specify exactly which subband and which quantity are required.

A particular subband is uniquely specified by its so-called subband ID, which is determined by the valley name of the subband and its subband index within this valley, separated by an underscore (_). For example, the subband ID of `Delta1_0` corresponds to the 0th subband of the `Delta1` valley.

Note:

Subband indexing starts from 0.

Therefore, a particular subband-based quantity is specified by giving the subband ID and then the quantity name, again separated by an underscore. For example, the name `Delta1_0_DistributionFunction` specifies the distribution function for the 0th subband of the `Delta1` valley.

In the following command sections, various subband-based quantities are described using names such as `<subbandID>_DistributionFunction`. Here, `<subbandID>` should be replaced by the subband ID such as `Delta1_0`.

BTESaveE

This command saves the solution of the subband-BTE performed over the channel coordinate–energy space.

Syntax

```
BTESaveE models=List tdrFile=String [slicedChannel=String]
```

Argument	Description	Default	Unit
`models`	Specifies a list of the models or quantities to save.	–	–
`tdrFile`	Sets the name of the TDR file to save.	–	–
`slicedChannel`	Sets the name of the sliced channel from which the quantities will be extracted.	First sliced channel defined	–

The available quantities that can be included in the list of `models` to save are:

Quantity	Description	Unit
`ChannelCoord`	Position along the channel; saved by default	µm
`Energy`	Total carrier energy; saved by default	eV
`<subbandID>_BackwardNetScatteringRate`	Net in-scatter minus out-scatter rate into backward-going states	s^{-1}
`<subbandID>_CurrentSpectrum`	Energy-resolved current spectrum used to compute the current	A/eV
`<subbandID>_DensitySpectrum`	Energy-resolved density spectrum used to compute the inversion density	$(eV*cm)^{-1}$
`<subbandID>_DistributionFunction`	Carrier distribution function used to compute the carrier density	1
`<subbandID>_DOS`	Density-of-states (DOS) per spin and direction	$(eV*cm)^{-1}$
`<subbandID>_ForwardNetScatteringRate`	Net in-scatter minus out-scatter rate into forward-going states	s^{-1}
`<subbandID>_SubbandEnergy`	Zero contour gives subband energy along the channel coordinate	1

Description

The solution of the subband-BTE is performed over the channel coordinate–energy space. A few major quantities are solved or computed on this grid on a per-subband basis. The `BTESaveE` command allows you to save a 2D TDR file over the channel coordinate–energy space for these quantities.

For quantities other than `SubbandEnergy`, you can use the `subbandID` to save a quantity per subband, per valley, or as the total value summed over all subbands:

- To save a quantity per subband, the `subbandID` must contain only the subband name.

  For example, to `save the DensitySpectrum for the Delta1_0` subband, specify the quantity `Delta1_0_DensitySpectrum`.

- To save a quantity per valley, the `subbandID` must contain only the valley name.

  For example, to save the sum of the `DensitySpectrum` for all subbands in the `Delta1` valley, specify the quantity `Delta1_DensitySpectrum`.

- To save the total value of a quantity, do not specify the `subbandID`.

  For example, to save the sum of the `DensitySpectrum` over all subbands, specify the quantity `DensitySpectrum`.

Examples

Save a TDR file named `FieldsOver_CC_E.tdr` containing the distribution function and the current spectrum for the `Delta1_0` subband:

```
BTESaveE tdrFile=FieldsOver_CC_E.tdr \
   models=[list Delta1_0_DistributionFunction Delta1_0_CurrentSpectrum]
```

BTESaveK

This command saves a 2D TDR file over the channel coordinate–*k*-space for the specified quantities.

When the solution of the subband-BTE is performed over the channel coordinate–energy space, it might be helpful to visualize some quantities over the channel coordinate–*k*-space instead, in particular, the distribution function. The `BTESaveK` command saves a 2D TDR file over channel coordinate–*k*-space for the specified quantities. The `Nk` and `Kmax` arguments are provided to specify the *k*-grid that should be used.

Syntax

```
BTESaveK models=List tdrFile=String \
   [Kmax=Double] [Nk=Integer] [slicedChannel=String]
```

Argument	Description	Default	Unit
`models`	Specifies a list of the models or quantities to save.	–	–
`tdrFile`	Sets the name of the TDR file to save.	–	–
`Kmax`	The *k*-grid is created for points between `-Kmax` and `Kmax`.	0.3	$2\pi/a_0$
`Nk`	Sets the number of *k*-points to use in the *k*-grid.	101	1
`slicedChannel`	Sets the name of the sliced channel from which the quantities will be extracted.	First sliced channel defined	–

The available quantities that can be included in the list of `models` to save are:

Quantity	Description	Unit
`ChannelCoord`	Position along the channel; saved by default	µm
`k`	The *k*-value along the *k*-grid; saved by default	$2\pi/a_0$
`<subbandID>_Dispersion`	Subband dispersion	eV
`<subbandID>_DistributionFunction`	Carrier distribution function used to compute the carrier density	1

Examples

Save a TDR file named `FieldsOver_CC_K.tdr` containing the distribution function for the `Delta1_0` subband over channel coordinate–*k*-space (the *k*-grid that is saved extends from $-0.3*2\pi/a_0$ to $0.3*2\pi/a_0$ with 101 points):

```
BTESaveK tdrFile=FieldsOver_CC_K.tdr \
   models=[list Delta1_0_DistributionFunction] Nk=101 Kmax=0.3
```

BTESaveCC

This command saves quantities from the solution of the subband-BTE that depend only on the channel coordinate.

Several quantities from the solution of the subband-BTE depend only on the channel coordinate. These include the inversion charge, the current, and the carrier velocity. These quantities can be saved on a per-subband basis. In addition to the quantities mentioned, the total value from all subbands can be saved.

Syntax

```
BTESaveCC models=List tdrFile=String [slicedChannel=String]
```

Argument	Description	Default	Unit
`models`	Specifies a list of the models or quantities to save.	–	–
`tdrFile`	Sets the name of the TDR file to save.	–	–
`slicedChannel`	Sets the name of the sliced channel from which the quantities will be extracted.	First sliced channel defined	–

The available quantities that can be included in the list of `models` to save are:

Quantity	Description	Unit
`BackwardCurrentTotal`	Total current flowing in the backward direction	A
`ChannelCoord`	Position along the channel; saved by default	µm
`CurrentTotal`	Total current	A
`Diffusivity`	Diffusivity	cm^2/s
`DopingConcentration`	Integral of the doping concentration over each slice	cm^{-1}
`EnergyCurrentTotal`	Total energy current	W
`ForwardCurrentTotal`	Total current flowing in the forward direction	A
`LatticeTemperature`	Average temperature over one slice	K
`Mobility`	Mobility	cm^2/Vs
`NinvTotal`	Total inversion charge	cm^{-1}
`VelocityTotal`	Total carrier velocity	cm/s
`<subbandID>_BackwardCurrent`	Current flowing in the backward direction	A
`<subbandID>_Current`	Current	A
`<subbandID>_Efleq`	Local Fermi level for the RTA scattering model	eV
`<subbandID>_EnergyCurrent`	Energy current	W
`<subbandID>_ForwardCurrent`	Current flowing in the forward direction	A
`<subbandID>_Ninv`	Inversion charge	cm^{-1}
`<subbandID>_Occupancy`	Subband occupancy	1
`<subbandID>_SubbandEnergy`	Minimum subband energy	eV
`<subbandID>_Velocity`	Carrier velocity	cm/s

Examples

Save a TDR (xy) file named `FieldsOver_CC.tdr` containing the total inversion charge along the channel as well as the velocity in the 0th subband of the `Delta1` valley (since no sliced channel is specified, the first defined sliced channel is used automatically):

```
BTESaveCC tdrFile=FieldsOver_CC.tdr models=[list NinvTotal \
   Delta1_0_Velocity]
```

Saving Slice-Related Fields

Each 2D slice is like a 2D device with a 2D nonlocal area defined. Like a 2D device, real-space models over this 2D device can be saved as well as 1D *k*-space data similar to what is done for Schrödinger and mobility calculations on a single 2D device in Sentaurus Band Structure.

The model syntax is exactly the same. The key difference here is that a particular 2D slice from a sliced channel must be selected. A 2D slice has a unique *address* given by its sliced channel name and its channel coordinate. The user-specified `ChannelCoord` snaps to the nearest actual slice position.

SaveSlice

This command saves real-space models over the 2D slice, specified with the `slicedChannel` and `channelCoord` arguments, to a TDR file. The coordinates of the real-space mesh are saved in units of micrometer.

Syntax

```
SaveSlice channelCoord=Double models=List tdrFile=String \
   [slicedChannel=String]
```

Argument	Description	Default	Unit
`channelCoord`	Specifies the channel coordinate of the required slice. The specified value snaps to the nearest slice position.	–	µm
`models`	Specifies a list of the models or quantities to save. The usual set of real-space models over a 2D device can be specified.	–	–
`tdrFile`	Sets the name of the TDR file to save.	–	–
`slicedChannel`	Sets the name of the sliced channel from which the quantities will be extracted.	First sliced channel defined	–

Examples

Save a 2D TDR file named `slice.tdr` containing the mesh and specified models over the 2D slice located in the sliced channel named `channel1` and closest to the slice channel coordinate of 20.0e-3 µm:

```
SaveSlice tdrFile=slice.tdr slicedChannel=channel1 \
   channelCoord=20.0e-3 \
   models=[list ConductionBandEnergy Delta1_0_Wavefunction]
```

Two models are saved: the relaxed conduction band energy and the norm of the wavefunction for the `Delta1_0` subband.

SaveSliceK

This command saves a TDR (xy) file of *k*-space models over 1D *k*-space for the slice specified with the `slicedChannel` and `channelCoord` arguments. The 1D *k*-space coordinates are saved in units of $2\pi/a_0$. The *k*-space grid that is used is specified using the `Nk` and `Kmax` arguments.

Syntax

```
SaveSliceK channelCoord=Double models=List tdrFile=String \
   [Kmax=Double] [Nk=Integer] [slicedChannel=String]
```

Argument	Description	Default	Unit
`channelCoord`	Specifies the channel coordinate of the required slice. The specified value snaps to the nearest slice position.	–	µm
`models`	Specifies a list of the models or quantities to save. Only the dispersion can be saved.	–	–
`tdrFile`	Sets the name of the TDR file to save.	–	–
`Kmax`	The *k*-grid is created for points between `-Kmax` and `Kmax`.	0.3	$2\pi/a_0$
`Nk`	Sets the number of *k*-points to use in the *k*-grid.	101	1
`slicedChannel`	Sets the name of the sliced channel from which the quantities will be extracted.	First sliced channel defined	–

Examples

Save a TDR (xy) file named `slice_K.tdr` containing a 1D *k*-space grid and specified models over 1D *k*-space for the slice located in the sliced channel named `channel1` and closest to the slice channel coordinate of 20.0e-3 µm:

```
SaveSliceK tdrFile=slice_K.tdr slicedChannel=channel1 \
   channelCoord=20.0e-3 models=[list Delta1_0_Dispersion]
```

In this example, only the dispersion for the `Delta1_0` subband is saved. The default values for `Nk` and `Kmax` are used.

Part II: NEGF Solver

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

5

Introduction to the NEGF Solver

This chapter describes the features of the quantum transport equation solver of Sentaurus Device QTX and the physical models it uses to model ballistic and dissipative transport.

Features of the NEGF Solver

Sentaurus Device QTX features the quantum transport equation solver (hereafter, referred to as the *NEGF solver*) that takes into consideration quantum-mechanical effects such as confinement, source-to-drain tunneling, and coherence effects like resonant tunneling.

Note:

You can perform ballistic and dissipative simulations for 3D structures such as FinFETs and nanowires as well as ballistic and dissipative simulations for 2D structures such as ultrathin bodies, as well as ballistic and dissipative simulations for bulk-like 1D structures.

The NEGF solver solves the Schrödinger equation with open boundary conditions in either the wavefunction formalism or the non-equilibrium Green's function (NEGF) formalism. The carrier and current densities of the simulated nanostructures are obtained by self-consistently coupling the solution of the Schrödinger and Poisson equations until convergence is reached.

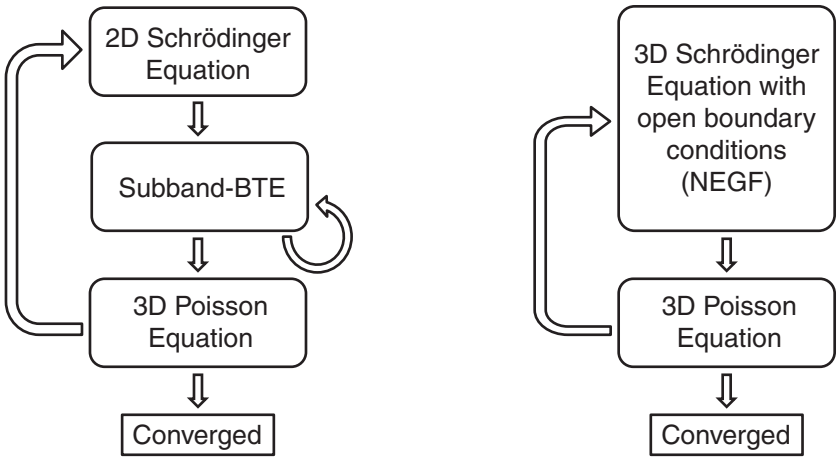
Similarities to Sentaurus Band Structure and the Subband-BTE Solver

The NEGF solver shares many commands and physical models with Sentaurus Band Structure and the Subband-BTE solver. Figure 9 on page 63 shows the simulation flows of the NEGF solver and the Subband-BTE solver.

The NEGF solver uses a quantum-mechanical description for the entire 3D system; whereas, the Subband-BTE solver uses a hybrid equation system combining 2D quantum mechanics with a 1D Boltzmann transport equation.

For descriptions of common commands, models, and parameters, see Part II of the *Sentaurus™ Device Monte Carlo User Guide*.

Figure 9 *Schematic of the iteration procedure, comparing (left) the Subband-BTE solver and (right) the NEGF solver*



Device Structure

The simulation of carrier transport using the solution of the NEGF solver can be performed for 1D, 2D, and 3D devices. The use of the NEGF solver is most appropriate for devices with a well-defined channel, in which geometric quantum confinement plays a strong role. This is typically the case in small FinFETs, nanowire devices, and ultrathin-body structures. The device structure is loaded from a TDR file by using the `LoadDevice` command.

Requirements and Restrictions on Device Structure and Mesh

The NEGF solver has some requirements and restrictions on the type of device structure and mesh that can be treated, namely:

- One-dimensional, 2D, and 3D structures can be simulated.
- The 2D or 3D mesh must be composed of all rectangles or cuboids, respectively (a tensor mesh). You can produce a tensor mesh using Sentaurus Mesh. For more information, see the *Sentaurus™ Mesh User Guide*.
- It is expected that the left and right ends of the structure are terminated by contacts.

Device Axes and Orientation

Depending on the dimensions of a structure, device axes must be set up correctly during structure generation:

- For *1D structures*, the device axis along the channel direction must be the *z-axis*.
- For *2D structures*, the device axis along the channel direction must be the *x-axis*. The confinement axis is the y-axis.
- For *3D structures*, the device axis along the channel direction must be the *z-axis*. The confinement axes must be the x-axis and the y-axis.

For both 2D and 3D structures, the orientation of the device relative to the crystallographic axes can be specified using the `xDirection` and `yDirection` arguments of the `Physics` command.

For 1D structures, the arguments `xDirection` and `surfaceOrientation` of the `Physics` command can be specified to define the orientation of the device relative to the crystallographic axes.

Contacts

It is expected that the left and right ends of the structure are terminated by contacts. Additional contacts, typically one gate contact, can be included in the structure as well. However, the current is computed only for the left and right contacts of a structure.

During the solution of the Poisson equation, a special Neumann boundary condition should be used at the left and right contacts to allow charge neutrality there to be maintained, even in the presence of quantum confinement. To activate this boundary condition, use the `Physics contact` command with `type=Neumann`. For example:

```
Physics contact=source type=Neumann
Physics contact=drain  type=Neumann
```

You can specify a lumped resistance, in units of ohm (Ω), for the left and right contacts of a nonlocal. For example, to specify a lumped resistance of 1kΩ for contacts named `source` and `drain`, specify:

```
Physics contact=source resist=1.0e3
Physics contact=drain  resist=1.0e3
```

Schottky Contacts

You also can introduce Schottky contacts for the left and right contacts. To allow Schottky contacts, during structure generation, you must place thin metal regions between the device

and the left and right contacts. (Metal regions are defined as regions where the material type is metal. The effective thickness of the region is not important, but it should consist of a couple of layers that will depend on the discretization in the transport direction, for example, thickness ~1 nm.)

Then, you can specify the Schottky contacts as:

```
Physics contact=source workfunction=4.5
Physics contact=drain  workfunction=4.5
```

or:

```
Physics contact=source schottkyBarrier=0.2
Physics contact=drain  schottkyBarrier=0.2
```

In the former case, the Schottky barrier is computed, for an n-type contact, as:

$$\phi_{\text{barrier}} = \phi_{\text{m}} - \chi_{\text{semi}} \tag{12}$$

and, for a p-type contact, as:

$$\phi_{\text{barrier}} = E_{\text{g}} - (\phi_{\text{m}} - \chi_{\text{semi}}) \tag{13}$$

where:

- ϕ_{m} is the metal workfunction.
- χ_{semi} is the electron affinity of the semiconductor.
- E_{g} is the band gap of the semiconductor.

In the later case, you set the Schottky barrier directly.

In addition, you have the option to define a virtual band-edge shift in the metal regions of the left and right contacts as proposed by [1]. The virtual band-edge shift is set in the `ValleyModel` definitions, for example, for the source metal region as:

```
Physics region=sourceMetal ValleyModel=ConstantEllipsoid name=Delta \
   useForEBulkDensity=1 degeneracy=2 \
   kl=[list 1 0 0] kt1=[list 0 1 0] kt2=[list 0 0 1] \
   ml=$ml mt1=$mt mt2=$mt virtualBandEdgeShift=-0.2
```

Note:

You can set the virtual band-edge shift for all valley models.

Nonlocal

First a nonlocal must be defined to solve the quantum transport equations along the channel of a device. The definition of a nonlocal region is performed exactly the same as described in the *Sentaurus™ Device Monte Carlo User Guide*.

The following example creates a nonlocal consisting of only the silicon region, named `si1`, of a silicon nanowire:

```
Math nonlocal name=channel1 regions=[list si1]
```

Specifying a Schrödinger Solver for a Nonlocal

Specifying a Schrödinger solver for a nonlocal is the same as in Sentaurus Band Structure. For example:

```
Physics nonlocal=channel1 eSchrodinger=Parabolic \
   valleys=[list Delta1 Delta2 Delta3]
```

Only these arguments are processed. All the remaining arguments are ignored because only the Hamiltonian of the 3D device is assembled here, and no band structures or corresponding wavefunctions are calculated.

Model and Parameter Specification

Similar to the specification of models and parameters for the subband and mobility calculations in Sentaurus Band Structure, for the NEGF solver, you can specify models and parameters on a material or region basis. The specified models and parameters are used in the nonlocal that contains the specified material or region.

Valley Models

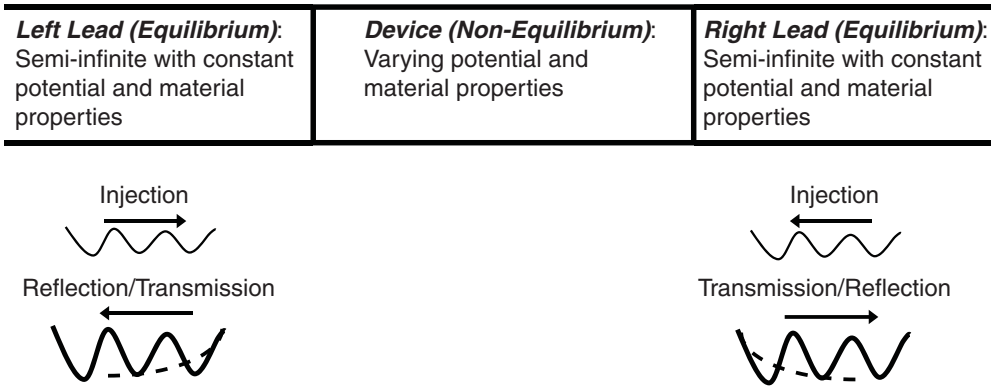
Valley models specify the band model and parameters to use when assembling the 1D, 2D, or 3D device Hamiltonian. The following valley models can be used for the NEGF solver:

- `ConstantEllipsoid`, `2kpEllipsoid`, and `2kpValley` can describe electrons.
- `3kpValley` and `6kpValley` can accurately describe holes.

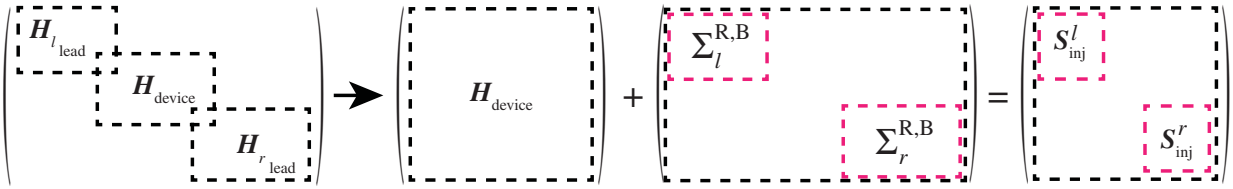
Quantum Transport

Figure 10 shows a schematic overview of quantum transport using the NEGF solver.

Figure 10 Device connected to two semi-infinite leads, contacts, or reservoirs



The systems of the equation can be closed by exploiting the properties of semi-infinite leads, which give rise to the retarded boundary self-energies, $\Sigma^{R,B}$, and the injected states, $\boldsymbol{S}_{\text{inj}}$.



Boundary Conditions

As previously stated, it is assumed that the leads are semi-infinite with constant potential and material properties. It is also assumed that the leads are in thermal equilibrium and, therefore, the electrons can be described by a Fermi–Dirac distribution. The semi-invariant nature of the contacts gives the asymptotic form of the solution of the Schrödinger equation, with left- and right-propagating Bloch waves. This fact allows you to close the equation system at the device boundaries by calculating $\Sigma^{R,B}$ and $\boldsymbol{S}_{\text{inj}}$ to define the coupling of the semi-infinite leads to the device [2][3][4].

The equation to compute the dispersion relation must be solved, but with exchanged input and output variables ($k \rightarrow E$):

$$((\boldsymbol{E} - \boldsymbol{H}_{ii}) - \boldsymbol{T}_{ii+1} e^{\pm ik(E)} - \boldsymbol{T}_{ii-1} e^{\mp ik(E)}) \varphi_i^{\pm}(E) = 0 \quad (14)$$

Here, $\boldsymbol{H}_{ii}$ and $\boldsymbol{T}_{ii+1}$ describe the constant diagonal and off-diagonal blocks in the leads, and the subscript *i* counts the unit cells along the transport direction.

Due to the translational invariant properties in the leads, $\boldsymbol{T}_{ii+1} = (\boldsymbol{T}_{ii-1})$ is valid. The variable $k(E)$ describes the wavevector divided by the discretization length in the transport

direction and is therefore unitless. The wavefunction vector $\varphi_i^{\pm}(E)$ describe the left (–) and right (+) propagating or decaying states.

Utilizing the Bloch theorem $\varphi_{i+1}(E) = \varphi_i(E)e^{ik(E)}$ together with the fact that only one of the equations in Equation 14 must be solved, the following generalized eigenvalue problem can be derived:

$$\begin{pmatrix} (\boldsymbol{E}-\boldsymbol{H}_{ii}) & -\boldsymbol{T}_{ii+1} \\ \mathbf{1} & \mathbf{0} \end{pmatrix}\begin{pmatrix} \varphi_i \\ \varphi_{i+1} \end{pmatrix} = e^{-ik(E)}\begin{pmatrix} \boldsymbol{T}_{ii-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{pmatrix}\begin{pmatrix} \varphi_i \\ \varphi_{i+1} \end{pmatrix} \tag{15}$$

Then, Equation 15 can be reformulated to obtain a normal eigenvalue problem [5]:

$$\begin{pmatrix} (\boldsymbol{E}-\boldsymbol{H}_{ii})-\boldsymbol{T}_{ii-1} & -\boldsymbol{T}_{ii+1} \\ \mathbf{1} & -\mathbf{1} \end{pmatrix}^{-1}\begin{pmatrix} \boldsymbol{T}_{ii-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{pmatrix}\begin{pmatrix} \varphi_i \\ \varphi_{i+1} \end{pmatrix} = \frac{1}{(e^{-ik(E)}-1)}\begin{pmatrix} \varphi_i \\ \varphi_{i+1} \end{pmatrix} \tag{16}$$

Equation 16 must be solved for the left contact and right contact. Finally, the expression for the boundary self-energy looks as follows:

$$\sum^{\mathrm{R,B}}(E) \sim (\boldsymbol{T}_{ii-1}\varphi_i^-)\cdot((\varphi_i^-(E))^{\dagger}\boldsymbol{T}_{ii+1}\varphi_i^-(E)e^{-ik(E)})^{-1}\cdot(\boldsymbol{T}_{ii-1}\varphi_i^-(E))^{\dagger} \tag{17}$$

where φ_i^- indicates that only states leaving the device (propagating or decaying) are needed. For the right-hand side of the linear equation system, you have:

$$\boldsymbol{S}_{\mathrm{inj}}(E) \sim [(\boldsymbol{T}_{ii-1}\varphi_i^{+,p}) - (\boldsymbol{T}_{ii-1}\varphi_i^-)\cdot((\varphi_i^-(E))^{\dagger}\boldsymbol{T}_{ii+1}\varphi_i^-(E)e^{-ik(E)})^{-1}\cdot((\varphi_i^-(E))^{\dagger}\boldsymbol{T}_{ii+1}\varphi_i^{+,p}(E)e^{ik^p(E)})] \cdot \frac{1}{\sqrt{dE/dk^p(E)}} \tag{18}$$

Here, $\varphi_i^{+,p}$ indicates the propagating waves traveling towards the injection direction (that is, states entering the device).

Schrödinger Equation With Open Boundary Conditions

The Schrödinger equation with open boundary conditions can be solved in the wavefunction formalism or the non-equilibrium Green's function (NEGF) formalism.

Wavefunction Formalism

In the wavefunction formalism, the scattering states of the central region (device region) are matched to the Bloch modes of the semi-infinite left and right leads [6]. Finally, a sparse linear system of equations must be solved:

$$(\boldsymbol{E}-\boldsymbol{H}_{\mathrm{device}}-\sum_l^{\mathrm{R,B}}(E)-\sum_r^{\mathrm{R,B}}(E))\phi(E) = \boldsymbol{S}_{\mathrm{inj}}(E) \tag{19}$$

where:

- The energy matrix *E* is diagonal and defines the energy value for which the device scattering states ϕ are computed.
- The device Hamiltonian $\boldsymbol{H}_{\text{device}}$ is discretized on a real-space grid using the finite volume method. Since you use an ordered (tensor) grid and nearest-neighbor coupling in the Hamiltonian, the discretized Hamiltonian has a block tri-diagonal structure.
- $\sum_l^{\text{R,B}}$ and $\sum_r^{\text{R,B}}$ are the boundary self-energies, describing the coupling to the leads. The matrices show nonzero elements only in the upper-left corner block ($\sum_l^{\text{R,B}}$) and in the lower-right corner block ($\sum_r^{\text{R,B}}$).
- The $\boldsymbol{S}_{\text{inj}}$ matrix describes the multiple right-hand sides, which model the injection from the left and right leads.

Then, the charge density can be calculated as:

$$n(\boldsymbol{r}) = \sum_{n \in \text{ leads}} \int \frac{dE}{2\pi} \left| \langle \boldsymbol{r} | \phi^n(E) \rangle \right|^2 \left(1 + e^{\frac{E - E_f^n}{kT}} \right)^{-1} \tag{20}$$

For the current, the expression is more complicated. For further explanations, see References on page 78.

Qualitatively, you can use either the scattering matrix approach to define a transmission based on the ratio between the amplitude of the injected state and the transmitted state, or the conservation of the probability current to derive an expression for the current.

Non-Equilibrium Green's Function Formalism

In the NEGF formalism, the Green's function is used to solve the Schrödinger equation with open boundary conditions [7]:

$$\begin{gathered}(\boldsymbol{E} - \boldsymbol{H}_{\text{device}}(\boldsymbol{r}_1)) G^{\text{R}}(E, \boldsymbol{r}_1, \boldsymbol{r}_2) - \int (\sum_l^{\text{R,B}}(E, \boldsymbol{r}_1, \boldsymbol{r}) + \sum_r^{\text{R,B}}(E, \boldsymbol{r}_1, \boldsymbol{r})) G^{\text{R}}(E, \boldsymbol{r}, \boldsymbol{r}_2) d\boldsymbol{r} = \delta(\boldsymbol{r}_1 - \boldsymbol{r}_2) \\ G^{<}_{l,r}(E, \boldsymbol{r}_1, \boldsymbol{r}_2) = \iint (G^{\text{R}}(E, \boldsymbol{r}_1, \boldsymbol{r}_3))(\sum_{l,r}^{<,\text{B}}(E, \boldsymbol{r}_3, \boldsymbol{r}_4))(G^{\text{R}}(E, \boldsymbol{r}_4, \boldsymbol{r}_2))^{\dagger} d\boldsymbol{r}_3 d\boldsymbol{r}_4\end{gathered} \tag{21}$$

Using the finite volume method for discretization, the equations in Equation 21 can be brought into a matrix form as:

$$(\boldsymbol{E} - \boldsymbol{H}_{\text{device}} - \sum_l^{\text{R,B}}(E) - \sum_r^{\text{R,B}}(E)) G^{\text{R}}(E) = \lambda \tag{22}$$

Equation 22 shows the main computational task in the NEGF formalism, which is the matrix inversion to compute the retarded Green's function. Furthermore, the injection is included in the lesser Green's function:

$$G^{<}_{l,r}(E) = \boldsymbol{G}^{\mathrm{R}}(E)(\sum\nolimits^{<,\mathrm{B}}_{l,r}(E))(\boldsymbol{G}^{\mathrm{R}}(E))^{\dagger}$$
$$\sum\nolimits^{<,\mathrm{B}}_{l,r}(E) = -\left(\sum\nolimits^{\mathrm{R,B}}_{l,r}(E) - (\sum\nolimits^{\mathrm{R,B}}_{l,r}(E))^{\dagger}\right) \tag{23}$$

In Equation 22, $\boldsymbol{G}^{\mathrm{R}}$ is the retarded Green's function matrix of the device, and λ describes the discretized delta function $\delta(\boldsymbol{r}_1 - \boldsymbol{r}_2)$. Equation 22 is solved by inverting a sparse tri-diagonal matrix.

To do this, the recursive Green's function (RGF) [8] algorithm is used. Finally, the charge density can be calculated as:

$$n(\boldsymbol{r}) = -i \sum_{n \in \text{ leads}} \int \frac{dE}{2\pi} diag\{G^{<}_{n}(E, \boldsymbol{r}, \boldsymbol{r}')\}\left(1 + e^{\frac{E - E^{n}_{f}}{kT}}\right)^{-1} \tag{24}$$

For the expression of the current, see References on page 78.

Qualitatively, the continuity equation leads to the following expression for the current in the NEGF formalism:

$$J(\boldsymbol{r}) \sim \frac{i}{2} \lim_{r \to r'} [\nu(\boldsymbol{r}) - \nu(\boldsymbol{r}')] \int \frac{dE}{2\pi} G^{<}(E, \boldsymbol{r}, \boldsymbol{r}') \tag{25}$$

The velocity $\nu(\boldsymbol{r})$ can also be expressed using commutator brackets as $\nu(\boldsymbol{r}) = -\frac{i}{\hbar}[\boldsymbol{r}, \boldsymbol{H}]$.

It has been shown in the expressions for the total electron density, in either the wavefunction formalism or the NEGF formalism, that you can distinguish which part is related to the left contact or right contact (coherence). This is possible only for ballistic simulations. If scattering is switched on, coherence is destroyed and you can no longer clearly assign an electron to either the left contact or the right contact.

Mode-Space Simulations

As already mentioned, the NEGF solver is based on a real-space grid and uses the finite volume method to discretize the Hamiltonians.

However, in addition to real-space simulations, mode-space simulations can be performed [9] in the NEGF formalism and the wavefunction formalism.

For ultrascaled structures such as small FinFETs or gate-all-around nanowire FETs, the sizes of matrices in the equation systems can be reduced significantly in the mode-space formulation without losing accuracy. Therefore, the computational burden also decreases dramatically.

Scattering in the Non-Equilibrium Green's Function Formalism

The NEGF formalism addresses the problem of dissipative quantum transport in a consistent and complete way [7][10]. Scattering processes can be introduced in a straightforward way through additional self-energies as follows:

$$\begin{aligned}
&(\boldsymbol{E} - \boldsymbol{H}_{\text{device}} - \sum^{\text{R,B}}(E) - \sum_1^{\text{R,scatt}}(E) - \sum_2^{\text{R,scatt}}(E))\boldsymbol{G}^{\text{R}}(E) = \lambda \\
&\boldsymbol{G}^{<}(E) = \boldsymbol{G}^{\text{R}}(E)(\sum^{<,\text{B}}(E) + \sum_1^{<,\text{scatt}}(E) + \sum_2^{<,\text{scatt}}(E))(\boldsymbol{G}^{\text{R}}(E))^{\dagger} \\
&\boldsymbol{G}^{>}(E) = \boldsymbol{G}^{\text{R}}(E)(\sum^{>,\text{B}}(E) + \sum_1^{>,\text{scatt}}(E) + \sum_2^{>,\text{scatt}}(E))(\boldsymbol{G}^{\text{R}}(E))^{\dagger}
\end{aligned} \tag{26}$$

where $\sum^{\text{R,B}}(E) = \sum_l^{\text{R,B}}(E) + \sum_r^{\text{R,B}}(E)$ describes the connection to the left and right leads as described in the previous section.

The so-called scattering self-energies $\sum_1^{<>,\text{scatt}}(E)$ and $\sum_2^{<>,\text{scatt}}(E)$ model scattering processes, such as electron–phonon interaction (1) and alloy disorder scattering (2). Because the NEGF formalism provides a systematic perturbation expansion, the scattering self-energies can describe the interaction processes up to the order of interest. Furthermore, in general, the scattering self-energies depend on the Green's function itself. For example:

$$\sum^{<>,\text{scatt}}(E) \sim \boldsymbol{G}^{<>}(E) \tag{27}$$

which gives rise to self-consistent computation loops, the so-called self-consistent Born iterations.

Finally, this means the equations at the beginning of this section are solved to evaluate $\boldsymbol{G}^{<>}(E)$, then $\boldsymbol{G}^{<>}(E)$ is used to compute $\sum^{<>,\text{scatt}}(E)$, and together with the relation:

$$\sum^{\text{R,scatt}}(E) \approx \frac{1}{2}(\sum^{>,\text{scatt}}(E) - \sum^{<,\text{scatt}}(E)) \tag{28}$$

the next iteration can be started until convergence is reached.

In the expression for $\sum^{\text{R,scatt}}(E)$, the principal integral term is neglected because it contributes only to an energy normalization and not to relaxation or phase breaking.

Quantum Transport in 2D Structures

For 2D structures, the z-direction is considered the free direction, which can be periodically extended. This means you can introduce a reciprocal lattice vector k_z to model the free direction and, at the same time, the real-space variable no longer defines a point in three dimensions. However, in a 2D plane, $\boldsymbol{r}_{xyz} \rightarrow \boldsymbol{r}_{xy}$ and $\boldsymbol{H}_{\text{device}}(\boldsymbol{r}_{xyz}) \rightarrow \boldsymbol{H}_{\text{device}}(\boldsymbol{r}_{xy}, k_z)$.

Then, the quantum transport equations also receive an additional k_z dependence. In the wavefunction formalism:

$$(\boldsymbol{E} - \boldsymbol{H}_{\text{device}}(k_z) - \sum_l^{\text{R,B}}(E, k_z) - \sum_r^{\text{R,B}}(E, k_z))\phi\ (E, k_z) = \boldsymbol{S}_{\text{inj}}(E, k_z) \tag{29}$$

In the Green's function formalism:

$$(\boldsymbol{E} - \boldsymbol{H}_{\text{device}}(k_z) - \sum_l^{\text{R,B}}(E, k_z) - \sum_r^{\text{R,B}}(E, k_z))\boldsymbol{G}^{\text{R}}\ (E, k_z) = \lambda \tag{30}$$

In general, this means these equations must be solved for every energy point and every k_z-point. This also leads to an additional k_z integration to compute the observable quantities. For example, the densities in the wavefunction formalism are calculated as:

$$n(\boldsymbol{r}) = \sum_{n\,\in\,\text{leads}} \int \frac{dk_z}{2\pi} \int \frac{dE}{2\pi} \left| \langle \boldsymbol{r} | \phi^n(E, k_z) \rangle \right|^2 \left(1 + e^{\frac{E - E_f^n}{kT}} \right)^{-1} \tag{31}$$

and the densities in the Green's function formalism are calculated as:

$$n(\boldsymbol{r}) = -i \sum_{n\,\in\,\text{leads}} \int \frac{dk_z}{2\pi} \int \frac{dE}{2\pi}\, \text{diag}\{ G_n^<(E, k_z, \boldsymbol{r}_{xy}, \boldsymbol{r}'_{xy}) \} \left(1 + e^{\frac{E - E_f^n}{kT}} \right)^{-1} \tag{32}$$

However, if an analytic band structure model like the effective mass approximation (EMA) is used to formulate the device Hamiltonian, the $k_z$treatment can be performed implicitly and the quantum transport equations to solve look the same as for the 3D case. Only the computation of the observable quantities must be adjusted. The expressions for the density, for example, look as follows:

$$\begin{aligned} n(\boldsymbol{r}) &= \sum_{n\,\in\,\text{leads}} \int \frac{dE}{2\pi} \left| \langle \boldsymbol{r} | \phi^n(E) \rangle \right|^2 \cdot \sqrt{\frac{m_z k_\text{B} T}{\hbar^2 2\pi}}\, \text{FDInt}_{-0.5}\left(-\frac{E - \mu}{k_\text{B} T} \right) \\ n(\boldsymbol{r}) &= -i \sum_{n\,\in\,\text{leads}} \int \frac{dE}{2\pi}\, \text{diag}\{ G_n^<(E, \boldsymbol{r}_{xy}, \boldsymbol{r}'_{xy}) \} \cdot \sqrt{\frac{m_z k_\text{B} T}{\hbar^2 2\pi}}\, \text{FDInt}_{-0.5}\left(-\frac{E - \mu}{k_\text{B} T} \right) \end{aligned} \tag{33}$$

where m_z is the effective mass in the z-direction, and $\text{FDInt}_{-0.5}$ is the Fermi–Dirac integral of order –1/2.

Note:

Here, the energy, E, no longer describes the total energy but the reduced energy in the transport direction.

Quantum Transport in 1D Structures

For 1D structures, the x- and y-directions are considered free directions, which can be extended periodically. This means you can introduce the 2D reciprocal lattice vector

$\boldsymbol{k}_t = (k_x, k_y)$ to model the free directions and, at the same time, the real-space variable no longer defines a point in three dimensions. However, on a 1D line, $r_{xyz} \rightarrow r_z$ and $H_{\text{device}}(r_{xyz}) \rightarrow H_{\text{device}}(r_z, k_x, k_y)$. Then, similar to Quantum Transport in 2D Structures, the quantum transport equations receive an additional k_x and k_y dependence, which leads to an additional k_x and k_y integration to compute the observable quantities. Because the extension from 2D to 1D structures is straightforward, the expressions for the quantum transport equations and observable quantities are omitted here for brevity.

Selecting an NEGF Solver

The NEGF solver can solve the ballistic quantum-transport equations using different formalisms:

- To use the wavefunction formalism, specify:

  ```
  Physics nonlocal=channel1 eNEGFSolver=BallisticWF
  ```

- To use the recursive Green's function (RGF) formalism, specify:

  ```
  Physics nonlocal=channel1 eNEGFSolver=BallisticGF
  ```

In general, you should use `BallisticGF` because it demonstrates better overall performance.

You can perform dissipative quantum-transport simulations only in the RGF formalism by specifying:

```
Physics nonlocal=channel1 eNEGFSolver=ScatteringGF
```

Scattering Models

The NEGF solver supports elastic and inelastic nonpolar-phonon scattering and elastic alloy scattering as well as Coulomb, surface roughness, and polar-optical phonon scattering. These scattering models can be defined in the same way as described in the documentation of Sentaurus Band Structure in the *Sentaurus™ Device Monte Carlo User Guide*.

For purely ballistic simulations, all scattering models must be removed by using the following command:

```
Physics material=all ScatteringModel removeAll
```

The elastic and inelastic nonpolar-phonon scattering models and the elastic alloy scattering model are local in the real-space variable due to the underlying approximations (isotropic matrix element). The Coulomb, surface roughness, and polar-optical phonon scattering models are, in principle, nonlocal in the real-space variable (anisotropic matrix element).

However, in addition for these scattering models, the diagonal approximation in the real-space variable is imposed to keep the computational burden manageable.

Furthermore, in the formulation of the scattering self-energies in mode space, it is assumed that the expression is also diagonal in the mode-space variable to keep the computational burden feasible.

The implemented scattering self-energies for elastic and inelastic nonpolar-phonon scattering and elastic alloy scattering take the following forms.

Note:

The Coulomb, surface roughness, and polar-optical phonon anisotropic scattering mechanisms are also implemented and available.

Acoustic Phonon Scattering

In real space, it can be written as:

$$\sum^{<>}(E,\boldsymbol{r},\boldsymbol{r}) = \frac{\mathrm{D}_{\mathrm{ac}}^2 k_{\mathrm{B}} T}{\rho u_l^2} G^{<>}(E,\boldsymbol{r},\boldsymbol{r}) \tag{34}$$

In mode space, the scattering self-energy can be written as:

$$\sum_{\mathrm{ii}}^{<>}(E,z,z) = \frac{\mathrm{D}_{\mathrm{ac}}^2 k_{\mathrm{B}} T}{\rho u_l^2} \sum_n \int d\boldsymbol{r}_\perp \left|\phi_n(z,\boldsymbol{r}_\perp)\right|^2 \left|\phi_i(z,\boldsymbol{r}_\perp)\right|^2 G_{nn}^{<>}(E,z,z) \tag{35}$$

where:

- D_{ac} is the effective deformation potential.
- ρ is the mass density.
- u_l is the longitudinal sound velocity.
- k_{B} and T are the Boltzmann constant and the temperature.
- $\phi_n(z,\boldsymbol{r}_\perp)$ is the mode-space basis function n at the real-space coordinate $(z,\boldsymbol{r}_\perp)$. The $\boldsymbol{r}_\perp$ indicates a point in the xy plane.

Inelastic Phonon Scattering

In real space, it can be written as:

$$\sum^{<>}(E,\boldsymbol{r},\boldsymbol{r}) = \frac{\hbar^2 D t K^2}{2\rho\hbar\omega}\left[N(\hbar\omega)G^{<>}(E\mp\hbar\omega,\boldsymbol{r},\boldsymbol{r}) + (N(\hbar\omega)+1)G^{<>}(E\pm\hbar\omega,\boldsymbol{r},\boldsymbol{r})\right] \tag{36}$$

In mode space, the scattering self-energy has the following form:

$$\sum_{ii}^{<>}(E,z,z) = \frac{\hbar^2 DtK^2}{2\rho\hbar\omega}\sum_n \int d\boldsymbol{r}_\perp \left|\phi_n(z,\boldsymbol{r}_\perp)\right|^2 \left|\phi_i(z,\boldsymbol{r}_\perp)\right|^2 [N(\hbar\omega)G_{nn}^{<>}(E\mp\hbar\omega,z,z) + (N(\hbar\omega)+1)G_{nn}^{<>}(E\pm\hbar\omega,z,z)] \quad (37)$$

where:

- *DtK* is the coupling constant.
- $\hbar\omega$ indicates the constant phonon energy.
- $N(\hbar\omega)$ describes the Bose–Einstein distribution function.

Alloy Disorder Scattering

The interaction with an alloy disorder is an elastic scattering process, and the scattering self-energies in real space and mode space are described as follows:

$$\sum^{<>}(E,\boldsymbol{r},\boldsymbol{r}) = U^2\frac{a_0}{8}x(1-x)G^{<>}(E,\boldsymbol{r},\boldsymbol{r})$$

$$\sum_{ii}^{<>}(E,z,z) = U^2\frac{a_0}{8}x(1-x)\sum_n \int d\boldsymbol{r}_\perp \left|\phi_n(z,\boldsymbol{r}_\perp)\right|^2 \left|\phi_i(z,\boldsymbol{r}_\perp)\right|^2 G_{nn}^{<>}(E,z,z) \quad (38)$$

where:

- *U* is the alloy potential.
- a_0 is the lattice constant.
- *x* indicates the mole fraction.

Self-Heating

Self-heating effects can be included in exactly the same way as for the Subband-BTE solver. For details, see Self-Heating on page 25.

Solving at One Bias

The solution of the NEGF solver at a specific bias is initiated using the `Solve` command. Similar to calculations using Sentaurus Band Structure, the bias on each contact is set using the `V(<contact>)` syntax, where `<contact>` refers to the contact name.

For example, to initiate a solve at specific biases on the source, drain, and gate contacts, use the following command:

```
Solve V(gate)=1.0 V(drain)=1.0 V(source)=0.0
```

If the bias on a particular contact is not specified explicitly in the `Solve` command, the previously specified bias is used. By default, the bias on all contacts is set to 0 at the start of the simulation.

Convergence Criteria

During the solution of the NEGF solver, the following equations are solved iteratively:

- Schrödinger equation with open boundary conditions in the wavefunction formalism or the NEGF formalism
- Poisson equation

The convergence criterion for the overall system for the Poisson equation is set by the `potentialUpdateTolerance` argument of the `Math` command (see Math on page 43). This argument, set in $k_B T/q$, determines the maximum-allowed change in the potential below which the Poisson equation is considered to have converged. For the NEGF solver, you should set `potentialUpdateTolerance` to a value of approximately 2.5e-2.

Simulating 2D Structures

The simulation flow for a 2D structure is the same as for 3D structures. You must load a 2D device structure by using the `LoadDevice` command.

Furthermore, in the case of k·p band structure models or scattering simulations, you must set the input parameter `kperpSum=1`, and you can use the parameters `Nkperp` and `kperpMax` to define the k_z grid. These parameters can be defined in conjunction with the `e(h)NEGFSolver=BallisticGF | ScatteringGF` parameter in the `Physics` section (see Physics for NEGF Solver on Nonlocal on page 93).

Note:

Two-dimensional structures can be simulated in the ballistic limit. Furthermore, the NEGF solver supports elastic and inelastic nonpolar-phonon scattering and elastic alloy scattering as well as Coulomb, surface roughness, and polar-optical phonon scattering. These scattering models can be defined in the same way as described in the documentation of Sentaurus Band Structure in the *Sentaurus™ Device Monte Carlo User Guide*.

Simulating 1D Structures

The simulation flow for a 1D structure is the same as for 2D or 3D structures. You must load a 1D device structure by using the `LoadDevice` command. Furthermore, in the case of k·p band structure models or scattering simulations, you must set the input parameter `kperpSum=1`, and you can use the arguments `Nkperp`, `kperpMax`, and `Nphiperp` to define the 2D *k*-space grid. These arguments can be defined in conjunction with `e(h)NEGFSolver=BallisticGF | BallisticWF | ScatteringGF` in the `Physics` section (see Physics for NEGF Solver on Nonlocal on page 93).

Note:

One-dimensional structures can be simulated in the ballistic limit. Furthermore, the NEGF solver supports elastic and inelastic nonpolar-phonon scattering and elastic alloy scattering. These scattering models can be defined in the same way as described in the documentation of Sentaurus Band Structure in the *Sentaurus™ Device Monte Carlo User Guide*.

Output

The primary output of the solution of the NEGF solver is the current, in ampere, flowing through the left and right contacts at the end of the structure. These current values are printed to the screen at the end of a `Solve` command, and the values also are stored in the bias log file.

In addition to the current, you can save several different TDR files with data from the solution of the NEGF solver.

2D TDR File With Fields Over Channel Coordinate–Energy Space

You can use the `NEGFSaveE` command to save a 2D TDR file with fields that are defined on the channel coordinate–energy grid. These fields include quantities such as the current spectrum and the density-of-states in each valley or in total (see NEGFSaveE on page 96).

TDR (XY) File With Fields Versus Channel Coordinate

You can use the `NEGFSaveCC` command to save a TDR (xy) file with fields versus the channel coordinate. The fields that can be saved include the total inversion charge, the total current, the total velocity, as well as these quantities per valley (see NEGFSaveCC on page 98).

TDR (XY) File With Fields Over 1D k-Space

You can use the `NEGFSaveContactEk` command to save the 1D k-space dispersion of the left contact and right contact computed by the Schrödinger solver (see NEGFSaveContactEk on page 99).

Parallelization

The NEGF solver supports message passing interface (MPI) parallelization but not multithreading.

An MPI parallelized job can be launched from Sentaurus Workbench or directly from the command line. For more information, see the *TCAD Parallelization Environment Setup User Guide* and the *Sentaurus™ Workbench User Guide*.

References

[1] J. Guo and M. S. Lundstrom, “A Computational Study of Thin-Body, Double-Gate, Schottky Barrier MOSFETs,” *IEEE Transactions on Electron Devices*, vol. 49, no. 11, pp. 1897–1902, 2002.

[2] M. Städele, B. R. Tuttle, and K. Hess, “Tunneling through ultrathin SiO_2 gate oxides from microscopic models,” *Journal of Applied Physics*, vol. 89, no. 1, pp. 348–363, 2001.

[3] M. P. López Sancho, J. M. López Sancho, and J. Rubio, “Highly convergent schemes for the calculation of bulk and surface Green functions,” *Journal of Physics F: Metal Physics*, vol. 15, no. 4, pp. 851–858, 1985.

[4] T. B. Boykin, “Generalized eigenproblem method for surface and interface states: The complex bands of GaAs and AlAs,” *Physical Review B*, vol. 54, no. 11, pp. 8107–8115, 1996.

[5] M. Luisier *et al*., “Atomistic simulation of nanowires in the $sp^3d^5s^*$ tight-binding formalism: From boundary conditions to strain calculations,” *Physical Review B*, vol. 74, p. 205323, November 2006.

[6] Z. Ren *et al*., “nanoMOS 2.5: A Two-Dimensional Simulator for Quantum Transport in Double-Gate MOSFETs,” *IEEE Transactions on Electron Devices*, vol. 50, no. 9, pp. 1914–1925, 2003.

[7] S. Datta, *Electronic Transport in Mesoscopic Systems*, Cambridge: Cambridge University Press, 1995.

[8] A. Svizhenko *et al*., “Two-dimensional quantum mechanical modeling of nanotransistors,” *Journal of Applied Physics*, vol. 91, no. 4, pp. 2343–2354, 2002.

[9] M. Luisier, A. Schenk, and W. Fichtner, “Quantum transport in two- and three-dimensional nanoscale transistors: Coupled mode effects in the nonequilibrium Green’s function formalism,” *Journal of Applied Physics*, vol. 100, no. 4, p. 043713, 2006.

[10] R. Lake *et al*., “Single and multiband modeling of quantum electron transport through layered semiconductor devices,” *Journal of Applied Physics*, vol. 81, no. 12, pp. 7845–7869, 1997.

6

Example: Getting Started

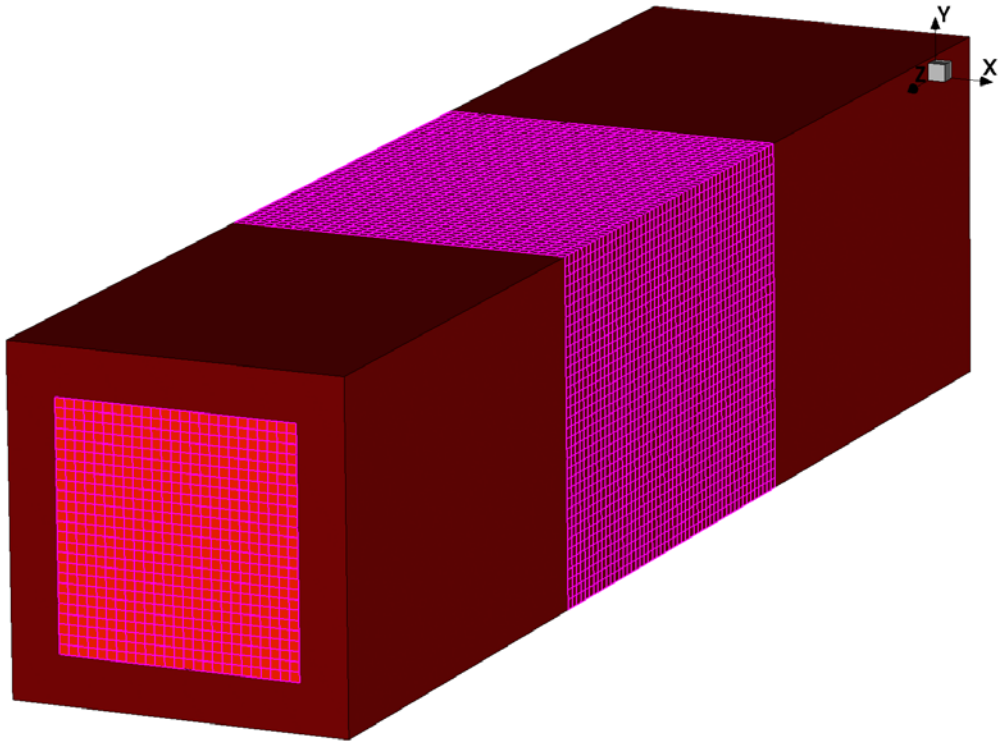
This chapter presents an example that uses the NEGF solver to compute an I_d–V_g curve for a silicon NMOS nanowire.

This chapter describes the command file for simulating ballistic quantum transport in a silicon nanowire device using the NEGF solver, including sections of the command file that load the device structure, set up the models, and perform the I_d–V_g simulation.

Device Structure

Figure 11 shows the device structure for this example. The device is a square silicon NMOS nanowire with a 5 nm × 5 nm cross section, a gate length of 10 nm, and a source and drain extension of length 10 nm. The source and drain are uniformly doped n-type to 2e20 cm^{-3}. This example computes an I_d–V_g curve at V_d = 0.6 V in the ballistic limit using the mode space option (`msSolver=0`).

Figure 11 Silicon nanowire device for the example



Loading the Device Structure

```
LoadDevice tdrFile=nanowire3D_0.tdr
```

The `LoadDevice` command loads the device structure from the `nanowire3D_0.tdr` file. After loading, the tool automatically sets up the default models and parameters in the silicon and oxide regions.

Setting Up the Valley Model and the Classical Solve

```
# Set orientation for <110> channel
Physics xDirection=[list -1 1 0] yDirection=[list 0 0 1]

# Remove all scattering models for ballistic simulation
Physics material=all ScatteringModel removeAll

# Remove all valley models
Physics material=all ValleyModel removeAll

# Set valley model
```

```
set valleys [list Delta1 Delta2 Delta3]
set ml 0.91
set mt 0.19

Physics material=Silicon ValleyModel=ConstantEllipsoid name=Delta1 \
   useForEBulkDensity=1 degeneracy=2 kl=[list 1 0 0] \
   kt1=[list 0 1 0] kt2=[list 0 0 1] ml=$ml mt1=$mt mt2=$mt

Physics material=Silicon ValleyModel=ConstantEllipsoid name=Delta2 \
   useForEBulkDensity=1 degeneracy=2 kl=[list 1 0 0] \
   kt1=[list 0 1 0] kt2=[list 0 0 1] ml=$mt mt1=$ml mt2=$mt

Physics material=Silicon ValleyModel=ConstantEllipsoid name=Delta3 \
   useForEBulkDensity=1 degeneracy=2 kl=[list 1 0 0] \
   kt1=[list 0 1 0] kt2=[list 0 0 1] ml=$mt mt1=$mt mt2=$ml

# Use Neumann boundary condition on source and drain
Physics contact=source type=Neumann
Physics contact=drain  type=Neumann

# Set workfunction
Physics contact=gate workfunction=4.2

# Simple model for hole density
Physics material=Silicon hBulkDensity=ConstantModel value=0.0

# Classical solve
Solve
```

These commands set up particular models and parameters, starting with setting the channel orientation. Then, all the scattering models are removed, so that a ballistic simulation can be performed. Next, all the valley models are removed and are set up from the beginning. The boundary condition type for the source and drain contacts is set to Neumann. The gate workfunction is set to 4.2 V.

Finally, a classical solve of only the Poisson equation is performed in equilibrium, that is, zero bias on all contacts. A *classical solve* means that the Schrödinger equation is not used. The resulting classical solution serves as the initial guess for the first solve using the Schrödinger equation.

Specifying the NEGF Solver

```
# Create a nonlocal over the Source, Channel, and Drain regions
Math nonlocal name=NL1 regions=[list Source Channel Drain]

# Set up parabolic Schrodinger on the nonlocal channel
Physics nonlocal=NL1 eSchrodinger=Parabolic \
   valleys=[list Delta1 Delta2 Delta3]
```

```
# Select NEGF solver and set mode-space parameters
Physics nonlocal=NL1 eNEGFSolver=BallisticGF msSolver=0 msNk=1 msNM=40
```

This section of the command file specifies the nonlocal, the Schrödinger solver, and the NEGF solver. The `Math nonlocal` command creates a nonlocal named `NL1` over the regions named `Source`, `Channel`, and `Drain`.

The first `Physics nonlocal` command specifies that a parabolic Schrödinger solver must be used on the nonlocal `NL1`. This command refers to the `Delta1`, `Delta2`, and `Delta3` valley models. For details about the `nonlocal` keyword, see the *Sentaurus™ Device Monte Carlo User Guide*.

The second `Physics nonlocal` command sets the NEGF solver to `BallisticGF` and specifies some mode-space parameters.

For the effective mass approximation (EMA), it is sufficient to have only one *k*-point (`k=0`) for the mode-space basis calculation; whereas, the number of modes depends on the size of the cross section and the energy range of interest. The larger the cross section or the wider the energy range of interest, which is determined by the bias conditions, the more modes are needed to describe the device characteristics correctly.

Performing the I_d–V_g Simulation

```
# Set some convergence parameters. It continues to next bias
#if not converged
Math potentialUpdateTolerance=1.0e-2 iterations=30 doOnFailure=0

# Solve at required drain bias and ramp gate using a Tcl foreach loop
set countVg 0
foreach Vg [list 0.0 0.1 0.2 0.3 0.4 0.5 0.6] {

   # Self-consistent Schrodinger-Poisson solve
   Solve V(source)=0.0 V(drain)=0.6 V(gate)=$Vg \
   logfile=QTX_NEGF_Quickstart

   # Add current to log file
   AddToLogFile name=Id value=[GetLast name=I(drain)]

   # Compute tunneling current
   set IdT [ComputeTunnelingCurrent]
   AddToLogFile name=IdTunneling value=$IdT

   # Save 3D TDR file
   Save tdrFile=QTX_NEGF_Quickstart_$countVg.tdr \
      models=[list DopingConcentration eDensity hDensity \
      ConductionBandEnergy]

   # Save fields over ChannelCoord-Energy
   NEGFSaveE tdrFile=QTX_NEGF_Quickstart_CC_E_$countVg.tdr \
```

```
        models=[list CurrentSpectrum DOS]

    # Save quantities over ChannelCoord
    NEGFSaveCC tdrFile=QTX_NEGF_Quickstart_CC_$countVg.tdr \
        models=[list NinvTotal CurrentTotal SubbandEnergy]

    # Save dispersion of the contacts
    NEGFSaveContactEk tdrFile=QTX_NEGF_Quickstart_Ek_$countVg.tdr \
        models=[list Delta1_0_DispersionSC Delta2_0_DispersionSC \
            Delta3_0_DispersionSC]
    incr countVg
}
```

This section of the command file computes the I_d–V_g curve. The `Math` command specifies the convergence tolerance for the Poisson equation. The number of allowed iterations for each bias is set to 30, and the `doOnFailure` argument is set such that the tool will continue to the next bias, even if the convergence criteria are not met.

The series of `Solve` commands write data to the default bias log file named `QTX_NEGF_Quickstart.plt`. Extra user-defined quantities named `Id` and `IdTunneling` are added to the bias log file to plot the I_d–V_g curve more easily in Sentaurus Visual.

Figure 12 shows the resulting I_d–V_g curve. Figure 13 shows the tunneling current flowing through the example device compared to the total current.

Figure 12 Ballistic I_d–V_g curve from the example

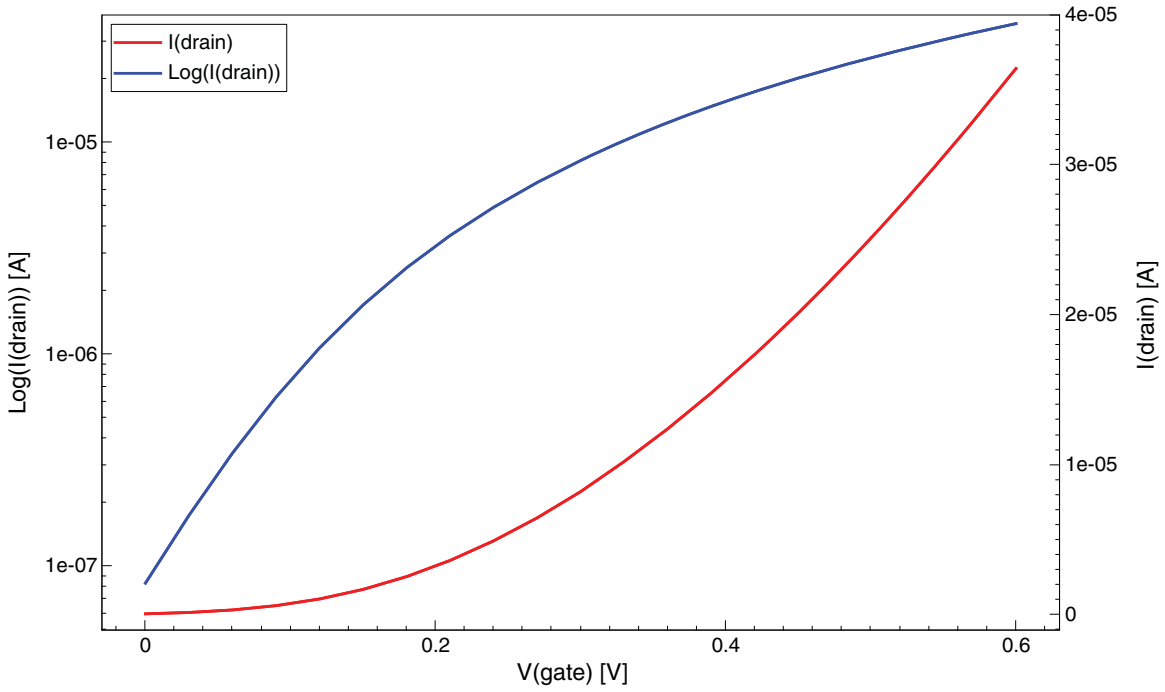
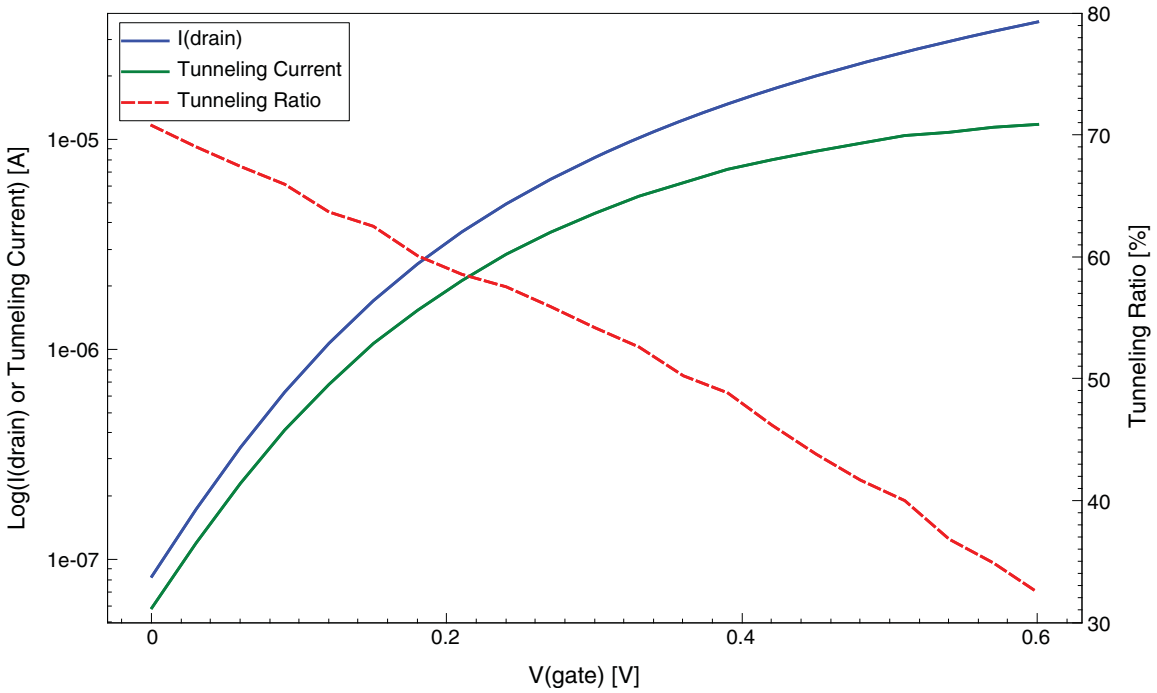


Figure 13 Portion of the total current that tunnels through the potential barrier under the gate

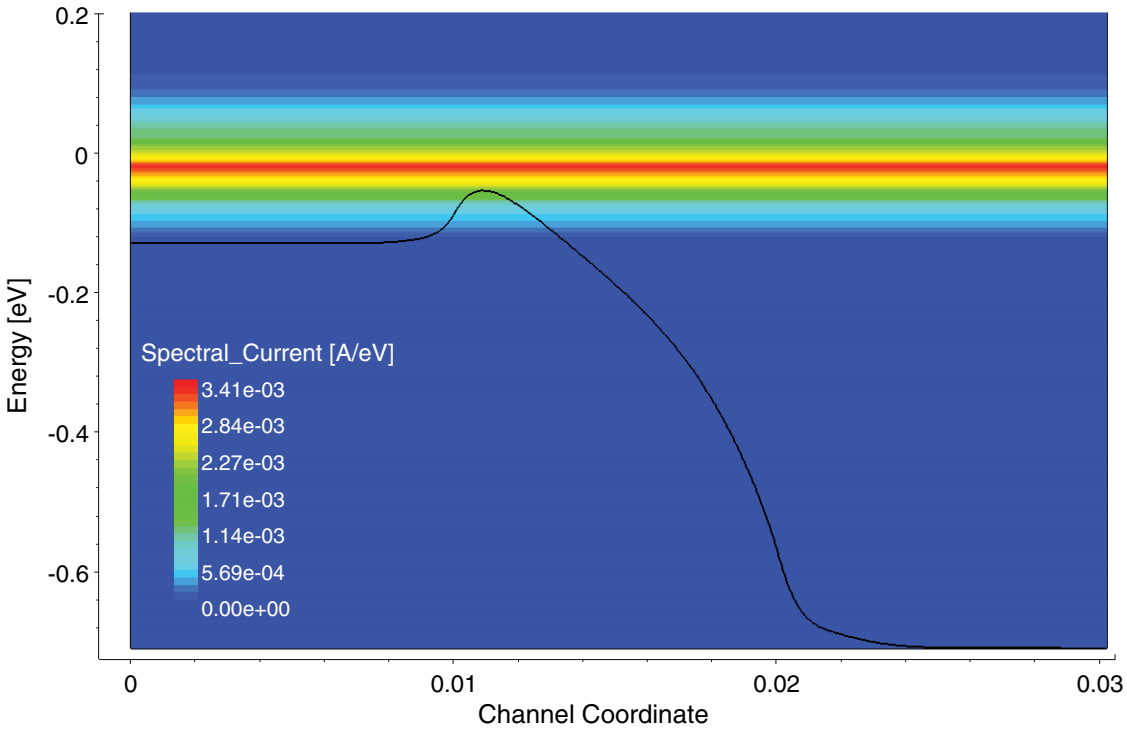


Saving TDR Files

For every `Solve`, the command file writes various TDR files with different types of data from the solution of the NEGF solver:

- The `Save` command writes a 3D TDR file with the 3D device structure and the selected models.
- The `NEGFSaveE` command writes a 2D TDR file over channel coordinate–energy space. In this example, the current spectrum and density-of-states are included. Figure 14 shows the spectral current density through the device.
- The `NEGFSaveCC` command saves a TDR (xy) file with the selected models versus the channel coordinate.
- The `NEGFSaveContactEk` command saves two TDR (xy) files with the dispersion of the left contact and right contact. Assuming the left contact and right contact of the device are named `source` and `drain`, the files `QTX_NEGF_Quickstart_Ek_source.tdr` and `QTX_NEGF_Quickstart_Ek_drain.tdr` are written. The files include the subband with index 0 for the `Delta1`, `Delta2`, and `Delta3` valleys.

Figure 14 Ballistic spectral current through the example device; solid black line indicates the potential distribution



7

Example: Dissipative Quantum Transport

This chapter presents an example that uses the NEGF solver to compute an I_d–V_{gs} curve for a silicon NMOS nanowire including electron–phonon interactions.

This chapter describes the command file for simulating dissipative quantum transport in a silicon nanowire device using the NEGF solver. The sections of the command file used to set up the electron–phonon interactions and the NEGF solver are described. The sections to load the device structure, to set up the valley models, and to perform the I_d–V_{gs} characteristics are exactly the same as those described in Chapter 6 on page 80.

Device Structure

The same device structure as in the previous example is used (see Device Structure on page 80).

Setting Up the Scattering Models

The scattering models can be defined in the same way as described in the documentation of Sentaurus Band Structure. For details about setting up scattering models, see the *Sentaurus™ Device Monte Carlo User Guide*. However, it is a good idea to remove all the scattering models and to start from the beginning, to ensure that the correct scattering models with the correct parameters are defined.

```
## Remove all scattering models to properly start from the beginning
Physics material=all ScatteringModel removeAll

########################################
## Scattering Models for Silicon
########################################

variable kb 8.617343e-5;     # [eV/K]  Boltzmann constant
variable massDensity 2.33;   # [g/cm3] Mass density
variable ul 9.05e5;          # [cm/s]  Speed of sound
variable Dac 14.6;           # [eV]    Deformation potential
```

```
## Acoustic Phonon
Physics material=Silicon ScatteringModel=ElasticAcousticPhonon \
   name="ElectronAC" \
   transitionType=Intravalley valleys=[list Delta1 Delta2 Delta3] \
   Dac=$Dac ul=$ul density=$massDensity

## g-type phonons
## Valley pairs for g-type intervalley scattering (equivalent)

variable gValleys [list {Delta1 Delta1} {Delta2 Delta2} {Delta3 \
   Delta3}]

## Parameter table for g-type phonon modes from Jungemann et al. [1]
##                     name  DtK     hbarOmega
lappend gParams [list ivg1  0.5e8   [expr $kb*140] ]
lappend gParams [list ivg2  0.8e8   [expr $kb*215] ]
lappend gParams [list ivg3 11.0e8   [expr $kb*720] ]
foreach paramSet $gParams {
   variable name      [lindex $paramSet 0]
   variable DtK       [lindex $paramSet 1]
   variable hbarOmega [lindex $paramSet 2]

   ## Abs
   Physics material=Silicon ScatteringModel=InelasticPhonon \
      name="${name}_ABS" transitionType=gIntervalley \
      inelasticType=ABS valleys=$gValleys \
      hbarOmega=$hbarOmega DtK=$DtK density=$massDensity

   ## Ems
   Physics material=Silicon ScatteringModel=InelasticPhonon \
      name="${name}_EMS" transitionType=gIntervalley \
      inelasticType=EMS valleys=$gValleys \
      hbarOmega=$hbarOmega DtK=$DtK density=$massDensity
}

## Valley pairs for f-type intervalley scattering (non-equivalent)
variable fValleys [list {Delta1 Delta2} {Delta1 Delta3} \
                        {Delta2 Delta1} {Delta2 Delta3} \
                        {Delta3 Delta1} {Delta3 Delta2} ]

## Parameter table for f-type phonons from Jungemann et al. [1]
##                     name  DtK     hbarOmega
lappend fParams [list ivf1 0.3e8  [expr $kb*220] ]
lappend fParams [list ivf2 2.0e8  [expr $kb*550] ]
lappend fParams [list ivf3 2.0e8  [expr $kb*685] ]
foreach paramSet $fParams {
   variable name      [lindex $paramSet 0]
   variable DtK       [lindex $paramSet 1]
   variable hbarOmega [lindex $paramSet 2]

   ## Abs
   Physics material=Silicon ScatteringModel=InelasticPhonon \
      name="${name}_ABS" transitionType=Intervalley \
```

```
      inelasticType=ABS valleys=$fValleys \
      hbarOmega=$hbarOmega DtK=$DtK density=$massDensity

## Ems
   Physics material=Silicon ScatteringModel=InelasticPhonon \
      name="${name}_EMS" transitionType=Intervalley \
      inelasticType=EMS valleys=$fValleys \
      hbarOmega=$hbarOmega DtK=$DtK density=$massDensity
}
```

This section of the command file sets up the electron–phonon scattering models. Interactions with elastic acoustic phonons as well as inelastic g-type and f-type phonons are considered.

Setting Up the Ballistic NEGF Solver and the Ballistic Solve

```
## Set up a ballistic solve first to have a good initial guess
Physics nonlocal=NL1 eNEGFSolver=BallisticGF \
   msSolver=0 msNk=1 msNM=40 msKmin=0.0

## Set some convergence parameters; continue to next bias
## if not converged
Math potentialUpdateTolerance=5.0e-3 potentialUpdateClamp=0.3 \
   iterations=20 doOnFailure=0

## Ballistic solve for a good initial guess
Solve V(source)=0.0 V(drain)=0.6 V(gate)=0.0 \
   logFile=QTX_NEGF_Quickstart_ballisticGuess.plt
```

This section specifies the ballistic NEGF solver and performs the ballistic solve for one specific V_{source}–V_{drain}–V_{gate} voltage point set, similar to the example in Chapter 6 on page 80. This section provides a good initial guess for the subsequent expensive scattering simulations.

Switching to the Scattering NEGF Solver

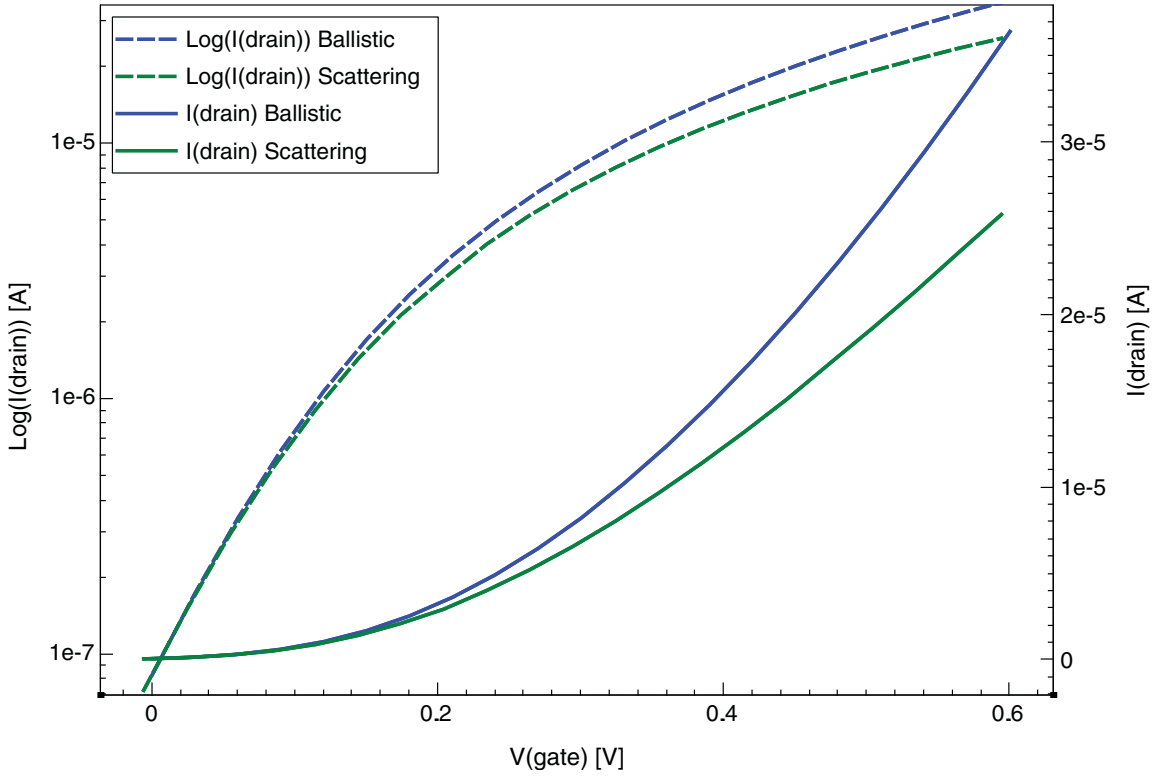
```
# Set up the scattering NEGF solver
Physics nonlocal=NL1 eNEGFSolver=ScatteringGF \
   msSolver=0 msNk=1 msNM=40 msKmin=0.0 scattIncInj=1 \
   scattBornIterMax=40 scattCurrTol=1.0e-2 scattDensTol=1.0e-3
```

In this section of the command file, the scattering NEGF solver is specified. The `Physics nonlocal` command sets the NEGF solver to `ScatteringGF` and defines some mode-space parameters together with some scattering parameters.

Results

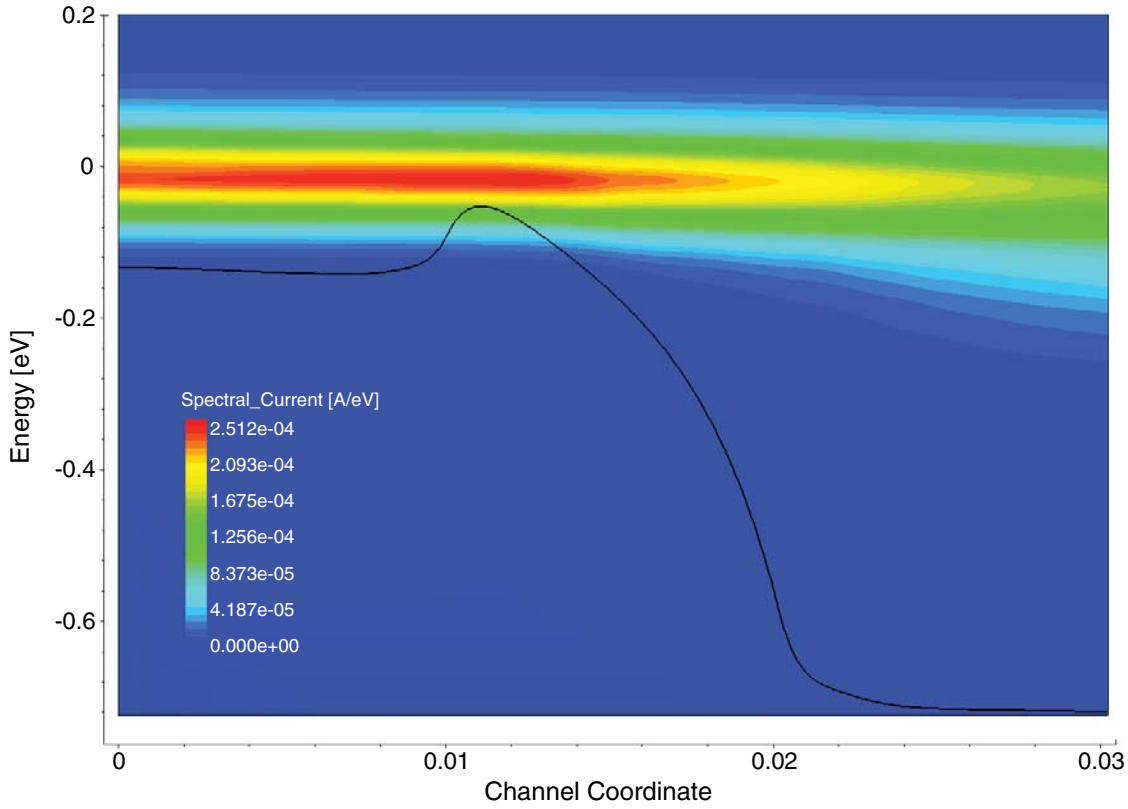
In Figure 15, you can see that electron–phonon interactions reduce the current as expected.

Figure 15 Comparison of I_d–V_{gs} characteristics between the ballistic and the scattering simulations



In Figure 16, you can clearly see that the electrons lose energy by phonon emission after the gate towards the drain.

Figure 16 Dissipative spectral current through the example device; black line indicates the potential distribution



References

[1] Chr. Jungemann, A. Emunds, and W. L. Engl, “Simulation of Linear and Nonlinear Electron Transport in Homogeneous Silicon Inversion Layers,” *Solid-State Electronics*, vol. 36, no. 11, pp. 1529–1540, 1993.

8

Commands of the NEGF Solver

This chapter summarizes the commands of the NEGF solver.

Physics Commands

This section describes the commands used to specify the NEGF solver.

Physics for Schrödinger Solver on Nonlocal

This command specifies the Schrödinger solver to use on a nonlocal. For NEGF simulations, the Schrödinger solver is used only to generate the 2D or 3D Hamiltonian of the device (nonlocal region). Therefore, only a few arguments are actually used here.

Syntax

```
Physics nonlocal=String eSchrodinger=String valleys=[list ...]
```

Argument	Description	Default	Unit
`nonlocal`	Sets the name of the nonlocal.	–	–
`eSchrodinger`	Sets the name of the Schrödinger solver for electrons to use on the nonlocal. The option is `Parabolic`.	–	–
`hSchrodinger`	Sets the name of the Schrödinger solver for holes to use on the nonlocal. Options are `3kp`, `6kp`, and `Parabolic`.	–	–
`valleys`	Specifies a list of valleys by name that the Schrödinger solver will use.	–	–

Physics for NEGF Solver on Nonlocal

This command specifies the NEGF solver to use on a nonlocal.

Syntax

```
Physics nonlocal=String eNEGFSolver=String msNK=Integer \
   msNM=Integer msSolver=Integer \
   [compensateForConfinement=Boolean]
   [contactEkKmax=Double] [contactEkNk=Integer] \
   [contactEkNM=Integer] [contactEkSolver=Boolean] \
   [dEin=Double] [dEout=Double] [dEsep=Double] \
   [Elimit=Double] [EminTail=Double] [Eoffset=Double] \
   [eta=Double] [kperpMax=Double] [kperpSum=Boolean] \
   [msKmax=Double] [msKmin=Double] [NEmax=Integer] \
   [Nkperp=Integer] [Nphiperp=Integer] \
   [scattBornIterMax=Integer] [scattCurrTol=Double] \
   [scattDensTol=Double] [scattIncInj=Boolean]
```

Argument	Description	Default	Unit
`nonlocal`	Sets the name of the nonlocal.	–	–
`eNEGFSolver`	Sets the name of the NEGF solver for electrons to use on the nonlocal. Options are: • `BallisticGF` or `BallisticWF` for ballistic simulations • `ScatteringGF` for scattering simulations	–	–
`hNEGFSolver`	Sets the name of the NEGF solver for holes to use on the nonlocal. Options are: • `BallisticGF` or `BallisticWF` for ballistic simulations • `ScatteringGF` for scattering simulations	–	–
`msNK`	Sets the number of *k*-points between `msKmin` and `msKmax`, uniformly distributed, for the mode-space basis calculation. The default value depends on whether the effective mass approximation (EMA) or k·p is used for the calculation.	1 (EMA) 5 (k·p)	–
`msNM`	Sets the number of modes for mode-space basis calculation.	20	–

Argument	Description	Default	Unit
`msSolver`	Specifies how to compute the eigenvalues and eigenstates for the mode-space basis calculation. Options are: • `-1`: None or real space • `0`: Arpack • `1`: Lapack	0	–
`compensateFor Confinement`	If set to 1, then the energy shift in the band structure computation, due to confinement, is compensated. This argument is particularly useful if strongly confined structures are simulated or if different valleys are set up based on more advanced band structure data (for example, density functional theory).	0	–
`contactEkKmax`	Sets the maximum value of the *k*-space. It must be in the range [0…π].	π	–
`contactEkNk`	Sets the number of *k*-points between 0 and `contactEkKmax`, uniformly distributed, for the calculation of the contact band structure.	51	–
`contactEkNM`	Sets the required number of subbands in the calculation of the contact band structure.	20	–
`contactEkSolver`	Specifies how to compute the eigenvalues for the contact band structure. Options are: • `0`: Arpack • `1`: Lapack	0	–
`dEin`	Sets the distance between two energy points immediately after the presence of a state with zero velocity in the contact band structure.	1.0e-3	eV
`dEout`	Sets the distance between two energy points in regions where no singularity is present.	2.0e-3	eV
`dEsep`	Sets the energy distance between a channel turn-on and the first upcoming discretization point.	5.0e-4	eV
`Elimit`	Sets the energy interval that is redefined after a mode is detected in the contact band structure. In this interval, the distance between two energy points changes incrementally from `dEin` to `dEout`, until `Elimit` is reached.	50.0e-3	eV

Argument	Description	Default	Unit
`EminTail`	Sets the energy interval that is considered below the lowest subband (taken from both the left contact and the right contact).	4.0e-3	eV
`Eoffset`	Sets the total energy interval that is considered. The energy vector goes then from `Emin`, which is the minimum energy of the system (automatically generated from the contact band structures and `EminTail`) to `Emax=Emin+Eoffset`.	0.5	eV
`eta`	Specifies the small imaginary part that is added to the energy values to ensure the retarded property of the Green's function.	0.0	eV
`kperpMax`	Defines the upper boundary of the transverse *k*-value.	0.5	$2\pi/a_0$
`kperpSum`	Activates (1) or deactivates (0) the explicit transverse *k*-summation.	0	–
`msKmax`	Sets the upper limit of the *k*-space for mode-space basis calculation.	$\pi/2$	–
`msKmin`	Sets the lower limit of the *k*-space for mode-space basis calculation.	0	–
`NEmax`	Sets the maximum number of calculated energy points.	2500	–
`Nkperp`	Defines the number of transverse *k*-points between 0 and `kperpMax`$*2\pi/a_0$, where a_0 is the lattice constant along the free direction.	21	–
`Nphiperp`	Sets the number of uniformly spaced points along the angular direction from 0 to 2π in the polar representation of the transverse *k*-grid. Only used if 1D structures are simulated.	11	–
`scattBornIterMax`	Defines the maximum number of self-consistent Born iterations between the Green's functions and the scattering self-energies.	25	–
`scattCurrTol`	Convergence criterion for the electron current. For example, a value of 5e-2 means that the difference between the current extracted at the beginning and at the end of the simulation domain must be less than 5%, and that current variations between two consecutive Born iterations must be less than 5%.	0.05	–

Argument	Description	Default	Unit
`scattDensTol`	Convergence criterion for the charge density in the self-consistent Born iterative process. For example, a value of 1e-2 means that charge variations between two consecutive Born iterations must be less than 1%.	0.01	–
`scattIncInj`	If set to 1, scattering is introduced in the semi-infinite reservoirs from which electrons are injected into the simulation domain. This removes artificial reflection at contact–device interfaces. If set to 0, the semi-infinite reservoirs are treated in the ballistic limit of transport.	0	–

Saving TDR Files

Several types of data and TDR files can be saved. Each type of TDR file is saved using a command specific to the type of data as described here.

NEGFSaveE

This command saves the solution of the quantum transport problem performed over the channel coordinate–energy space.

Syntax

```
NEGFSaveE models=List tdrFile=String [nonlocal=String]
```

Argument	Description	Default	Unit
`models`	Specifies a list of the models or quantities to save.	–	–
`tdrFile`	Sets the name of the TDR file to save.	–	–
`nonlocal`	Sets the name of the nonlocal from which the quantities will be extracted.	First nonlocal defined	–

The available quantities that can be included in the list of `models` to save are:

Quantity	Description	Unit
`ChannelCoord`	Position along the channel; saved by default	µm
`Energy`	Total carrier energy; saved by default	eV
`<valleyID>_CurrentSpectrum`	Energy-resolved current spectrum used to compute the current	A/eV
`<valleyID>_DensitySpectrum`	Energy-resolved density spectrum used to compute the inversion density	$(eV*cm)^{-1}$
`<valleyID>_DOS`	Energy-resolved density-of-states (DOS)	$(eV*cm)^{-1}$

Description

The solution of the quantum transport problem is performed over the channel coordinate–energy space. A few major quantities are solved or computed on this grid on a per-valley basis. The `NEGFSaveE` command allows you to save a 2D TDR file over the channel coordinate–energy space for these quantities.

You use the `valleyID` to save a quantity either per valley or as the total value summed over all valleys:

- To save a quantity per valley, the `valleyID` must contain the valley name.

  For example, to save `DensitySpectrum` for the `Delta1` valley, specify the quantity `Delta1_DensitySpectrum`.

- To save the total value of a quantity, omit the `valleyID`.

  For example, to save the sum of `DensitySpectrum` over all valleys, specify the quantity `DensitySpectrum`.

Examples

Save a TDR file named `FieldsOver_CC_E.tdr` containing the density and the current spectrum for the `Delta1` valley:

```
NEGFSaveE tdrFile=FieldsOver_CC_E.tdr \
   models=[list Delta1_DensitySpectrum Delta1_CurrentSpectrum]
```

NEGFSaveCC

This command saves the solution of the quantum transport problem averaged over the cross section. The quantities depend only on the channel coordinate.

Some quantities such as the inversion charge and the current can be saved over the channel coordinate. These quantities can be saved on a per-valley basis. In addition to the quantities mentioned, the total value from all valleys can be saved.

Syntax

```
NEGFSaveCC models=List tdrFile=String [nonlocal=String]
```

Argument	Description	Default	Unit
`models`	Specifies a list of the models or quantities to save.	–	–
`tdrFile`	Sets the name of the TDR file to save.	–	–
`nonlocal`	Sets the name of the nonlocal from which the quantities will be extracted.	First nonlocal defined	–

The available quantities that can be included in the list of `models` to save are:

Quantity	Description	Unit
`ChannelCoord`	Position along the channel; saved by default	µm
`CurrentTotal`	Total current	A
`EnergyCurrentTotal`	Total energy current	W
`LatticeTemperature`	Average temperature	K
`NinvTotal`	Total inversion density	cm^{-1}
`VelocityTotal`	Total carrier velocity	cm/s
`<valleyID>_Current`	Current	A
`<valleyID>_EnergyCurrent`	Energy current	W
`<valleyID>_Ninv`	Density	cm^{-1}

Quantity	Description	Unit
`<valleyID>_SubbandEnergy`	Subband energy along the channel coordinate	eV
`<valleyID>_Velocity`	Carrier velocity	cm/s

Examples

Save a TDR (xy) file named `FieldsOver_CC.tdr` containing the total inversion charge along the channel and the current of the `Delta1` valley (since no nonlocal is specified, the first defined nonlocal is used automatically):

```
NEGFSaveCC tdrFile=FieldsOver_CC.tdr \
   models=[list NinvTotal Delta1_Current]
```

NEGFSaveContactEk

This command saves the band structure of the contacts.

Syntax

```
NEGFSaveContactEk models=List tdrFile=String [nonlocal=String]
```

Argument	Description	Default	Unit
`models`	Specifies a list of contact dispersion models prepended by the valley name and the subband index. Valid models are: • `Dispersion` calculates the dispersion with zero potential in the computation. • `DispersionSC` includes the self-consistent potential in the computation.	–	–
`tdrFile`	Sets the main name of the TDR file to save. The name of the contact is added automatically to the end of the specified TDR file.	–	–
`nonlocal`	Sets the name of the nonlocal from which the quantities will be extracted.	First nonlocal defined	–

Examples

Assuming the two contacts at the very left and right of the device are named `source` and `drain`, write two TDR files (`ContactEk_source.tdr` and `ContactEk_drain.tdr`):

```
NEGFSaveContactEk tdrFile=ContactEk.tdr nonlocal=NL1 \
   models=[list Delta1_0_Dispersion Delta1_0_DispersionSC]
```

Both files include the subband with index 0 for the `Delta1` valley, one with zero potential (`Delta1_0_Dispersion`) and one including the self-consistent potential (`Delta1_0_DispersionSC`).

ComputeTunnelingCurrent

In addition to the previously described TDR output, you can compute the tunneling current, which is the portion of the total current that flows below the top of the barrier potential energy value, and add it to the current bias log file as:

```
set IdT [ComputeTunnelingCurrent]
AddToLogFile name=IdTunneling value=$IdT
```

The first command computes the tunneling current and stores it in the `IdT` variable. In the second command, the value given in `$IdT` is added to the field `IdTunneling` in the bias log file.