Solver

Drift-Diffusion Simulator User Guide

SemiVi LLC

Zollikon, Switzerland.

November 2, 2025

Contents

Chapter 1

Introduction

The DD solver provided by SemiVi can simulate the semiconductor devices by using drift-diffusion formalism. Also, temperature equations can also be coupled with the DD equations to model heat generation and transport during device operation. Additionally, the circuit-system composed of the semiconductor devices and the external circuit components is also solved coupled with the DD equations to simulate the device operation. It takes a device structure and the DD solver config file as an input and performs the simulations as specified in the config file. Static, quasi-stationary, transient simulations, as well as small-signal analysis can be performed with the DD solver.

The simulations generate a '.csv' file containing voltages and currents at each of the contacts. Also, spatial distribution of various physical quantities such as carrier densities, electrostatic potential are stored at various time points during the simulation.

1.1 Features

The DD solver has the following features.

- The DD solver can perform a quasi-stationary ramp, small-signal analysis at a fixed DC bias, and a transient analysis.

- Coupled Poisson, electron and hole continuity equations are solved in the DD solver for the above mentioned analyses.
- Coupled electro-thermal simulations can also be performed in the DD solver by solving temperature equation coupled with the continuity and Poisson equation.
- The circuit-system consisting of the semiconductor devices and the external components can be solved coupled with drift-diffusion equations.
- Physics based models are available in the DD solver to mimic various physical processes in semiconductors.
- The model parameters can be edited in the material config file, thus allowing the models to be calibrated to the experimental data.

1.2 Installation

SemiVi currently supports software installation on various Linux distributions. The software installer is available in Debian package (*.deb file) and in RPM format (*.rpm file).

Note, that *if you have downloaded mkl version of the DD solver*, the following package needs to be installed manually by you before installing the circuit solver from the installer package.

- Intel math kernel libraries (released in 2020 or later), which include distributions of open-mp, pardiso, etc. specific for Intel processors.

The DD solver sources mkl functions from the above installation. These functions can offer speed-up in the calculations on Intel processors. The mkl package can be downloaded from Intel website.

If the DD solver without mkl-acceleration is downloaded, then installation of the above package is not necessary.

Once `Intel math kernel libraries` are installed, download the installer on the local machine. The installer file named `ddsolver_amd64.deb` will appear in the `Downloads` directory. Go to the directory using `cd`

command. Use the following command to install the `DDSolver` from the installer.

```
>> sudo apt install ./ddsolver_amd64.deb
```

Alternately, one may use `dpkg` to install the software and use `apt` to install missing dependencies as follows.

```
>> sudo dpkg -i ./ddsolver_amd64.deb
>> sudo apt install -f
```

You need to have `root` access to install the software on your machine.

1.3 Licensing

Two types of licenses can be purchased for SemiVi DD solver.

Node-locked licenses enable unlimited number of *simultaneous* executions of the DD solver on the client machine. The node-locked license limits the usage of the DD solver only to the machine on which the license is activated.

With *server licenses*, the DD solver can be run on any of the machines in the client organization on which the server license is activated. However, only the specified number of *simultaneous* executions are possibie at a time.

1.3.1 Purchasing the licenses

The clients can place order for any of the above licenses on SemiVi website (https://www.semivi.ch/sales) or by contacting our salesperson.

We will process the request and send the license files by email. The license files need to be activated on the desired machines using the license key which is emailed separately using the following command.

1.3.2 Installation of SemiVi-activator

The license file must be activated on the desired computer before use. For that purpose, download the installer `semivila_amd64.deb` file on the local machine and install it as follows.

```
>> sudo apt install ./semivila_amd64.deb
```

1.3.3 License activation

To activate the license file, please run the following command.

```
>> semivila -a File.lic <Server|NodeLocked>License.lic\\
```

Replace `File.lic` with the your license file, and use appropriate name for the activated file. You will be prompted to input the 16 digit license key. A successful activate of the license file will generate the activated license file. Copy the activated license file to the `/opt/semivi/licenses/` folder and rename it to `ServerLicense.lic` or `NodeLockedLicense.lic` for server and node-locked licenses respectively. If you have more than one license files, please delete the older expired license files. If you wish to keep more than one active license files, you can also name the license files as `<i>NodeLockedLicense.lic` where `<i>` could be from 0 to 49. For ex. `49NodeLockedLicense.lic` or `49ServerLicense.lic`. The program will read the license files and lock the first available license. All the target users must have read rights on the license file.

User-guides of all the software provided by SemiVi are stored at the location `/opt/semivi/userguides/`.

Tutorials of all the software provided by SemiVi are stored at the location `/opt/semivi/tutorials/ddsolver`.

Chapter 2

Device Simulator Config File

DD solver software reads various inputs from DD solver configuration file and performs DD simulations on the device structure. The program can be executed using the following command –

```
>> DDSolver ddsolver FinFET_dev.cfg
```

In the above command, the word after `DDSolver` is the name of the program to be executed (in this case – `ddsolver`). The program name is followed by the configuration file name (in this case – `FinFET_dev.cfg`).

2.1 DD Solver Config File

A sample configuration file of the DDSolver is provided below.

```
File: {
  Device = "FinFET_str.cfg";
  Out = "FinFET";
}
```

```
Contacts: {
  gate: {Voltage = 0.0;  WorkFunction = 4.8; Type = ["Insulator"]; }
  source: {Voltage = 0.0; Type = ["Semiconductor"]; }
  drain: {Voltage = 0.0; Type = ["Semiconductor"]; }
}

Physics: {
  BandStructure: {
    CarrierDistribution: { ApproximateFermi: []; }
  }
  Mobility: {
    DopingDep: { Masetti: []; }
  }
  Recombination: {
    SRHRecombination = ["DopingDep"];
  }
}

Physics*Region*RegSi: {
  Mobility: {
    DopingDep: {
      Masetti: [];
    }
    FieldDep: {
      Lombardi: [];
    }
  }
  Recombination: {
    SRHRecombination = ["DopingDep"];
  }
}

Math:
{
  IterationsFC = 40;
  InnerIterationsFC = 10;
  IterationsSC = 40;
```

```
  UndampedIterations = 0;
  FCSolverTolerance = 1.;
  SCSolverTolerance = 1.;

  SolverSettings = [
            "InterpolateElecPotential",
            "InterpolateElecFermi"
            ,"InterpolateHoleFermi"
            ];
}

Plot:
{
  Quantities = ["ElectronDensity",
                "AbsElectricField",
                "ElectrostaticPotential",
                "TotalRecombination",
                "ElectronMobility",
                "AbsElectronCurrent",
                "AbsHoleCurrent"];
}

Solve:
{
  Static*Poisson: {
    Coupled: ["Poisson"];
    PlotTime = [0.];
  }

  Quasistationary*Drain: {
    initstep = 1E-3; minstep = 1E-5; maxstep = 0.1; incr = 1.35; decr
    Ramp: {
      Voltage*drain = 1.;
    }
    Coupled: ["Poisson","Electron"]
    PlotTime = [0., 0.5, 1.0];
    Math: {
      IterationsFC = 40;
```

```
      FCSolverTolerance = 1.;
    }
  }

  Quasistationary*Gate: {
    initstep = 1E-3; minstep = 1E-5; maxstep = 0.1; incr = 1.35; dec
    Ramp: {
      Voltage*gate = 1.0;
    }
    Coupled: ["Poisson","Electron"]
    Math: {
      IterationsFC = 40;
      FCSolverTolerance = 1.;
    }
    PlotTime = [0., 0.05, 0.5, 1.0];
  }
}
```

The above config file is composed of various sections which define the math settings, solve, physics, contacts, the quantities to be plotted, etc. In the config file, the string before '*' gives the section type, whereas the string after '*' specifies the section name. Various keywords in each of the section and their functionality is shortly described below.

2.2 File section

The keyword `Device` provides the file name from which the device structure is created. Internally, the file is processed differently according to its extension.

- If the file extension is "str.cfg", the file is processed as an input file for the tensor mesh generation.
- If the file extension is "str.h5", The file is read as hdf5 file generated by the structure and mesh generator. Note, that DD solver treats mesh created using external meshing programs such as *Triangle* or *TetGen* in the same way as internal quad-tree/Oct-tree based mesh.

The keyword `Out` sets the prefix to the output file name. In this case, the output files will be called 'FinFET_<id>_dev.xdmf' and 'FinFET_<id>_dev.h5'.

2.3 Contacts section

Properties and voltages of all the electrical contacts are initialized in this section. Every electrical contact defined in the structure file must be listed in the section. In this case, the contacts 'gate', 'source', and 'drain' have been listed. Each of them have a group of entries in curly brackets {...} in which initial voltage, contact resistance, and work-function are specified. Also, the keyword `Type` sets a list of strings which set the contact type. 'gate' is listed as an insulator contact while 'source' and 'drain' are semiconductor ohmic contacts.

2.4 Physics section

Region-specific, material-specific, or device-specific physical models can be listed in different physics sections of the config file. There can be only one global `Physics` section which lists the models active throughout the device. The section named 'Physics*Material*<mat>' lists models active in all the regions of material <mat>. Whereas, the section named 'Physics*Region*<reg>' lists models active in the region named <reg> in the device. In selecting active models in a given region, region-wise physics section gets precedence over material Physics section, whereas material physics section gets precedence over global physics section.

For convenience, bulk semiconductor models specified in the file have been separated into four sub-sections.

1. `BandStructure:` Use `ApproximateFermi` formula for calculating `CarrierDistribution`.

2. `Mobility:` Use `Masetti` formula to model doping dependence (`DopingDep`) of bulk mobility. Also, use `Lombardi` model for mobility degradation at the interfaces.

3. `Recombination:` Use `SRHRecombination` model together with doping dependent carrier lifetimes (`DopingDep`).

Multiple models can be specified in each section. If they are not conflicting with each other, they are combined together. Else, any one of the models gets precedence over the other.

2.5 Math section

Various parameters required for DD simulations are listed in `Math` section. Keywords `IterationsFC` and `IterationsSC` respectively, set maximum iterations of coupled solver and self-consistent solver. Similarly, `FCSolverTolerance` and `SCSolverTolerance` set tolerance of coupled solver and self-consistent solver, respectively.

`SolverSettings` gives a list of comma separated strings which specify what type of calculations are to be performed. For example, `InterpolateElecPotential`, `InterpolateElecFermi`, and `InterpolateHoleFerm` specifies that electrostatic potential, electron Fermi energy, and hole Fermi energy for the next bias point is set by extrapolating the respective quantities from current and previous bias point.

Parameters defined in global math section are applied to all the ramps in `solve` section. `Math` section can also be defined in each ramp in `Solve` section.

2.6 Plot section

A list of physical quantities to be stored is specified with the keyword `Quantities`. These physical quantities are stored at each bias point provided by `PlotTime` keyword in `Solve` section. They are stored in a hdf5 file '<out>_<id>_dev.h5', where <out> is the prefix provided in `File` section and <id> is the bias point number. A xdmf script file '<out>_<id>_dev.xdmf' is also saved for visualizing the above quantities in paraview.

2.7 Solve section

`Solve` section described the types of bias (or other) ramps to be performed in the same order as listed in the section. Three types of bias ramps, which are `Static`, `Quasistationary`, and `Transient` are available in DD solver.

`Static` simulation is performed at existing bias point, whereas `Quasistationary` simulation is performed at all the bias points starting from existing bias point ending at the bias point set by the `Ramp` group in `Quasistationary` simulation. In this case, `Static` simulation named 'Poisson' is performed at an initial bias point $V_G = 0V$, $V_D = 0V$, and $V_S = 0V$. After that, `Quasistationary` simulation named 'Drain' is performed in which V_D is ramped from 0V to 1.0V. This is specified in `Ramp` group by specifying final voltage by `Voltage*<Cont>` command. This is followed by a `Quasistationary` simulation named 'Gate' is performed in which V_G is ramped from 0V to 1.0V. Since the drain voltage is not specified in 'Gate' ramp, it is kept fixed at its value from the earlier 'Drain' ramp (1.0V).

`Coupled` simulation lists all the solution variables which are to be calculated using the coupled solver. In this case, `Poisson` and `Electron` specify that Poisson equation and electron continuity equation are solved using the coupled solver.

`PlotTime` specifies a list of time points at which values of all the listed physical quantities listed in `Plot` section are to be stored. Starting time of each quasi-stationary ramp is set to 0.0 and end time is 1.0.

`Math` section defined in each of the ramps, sets `Math` parameters specific for that specific ramp. If `Math` section is not specified in the given ramp, then the parameter values set in the global `Math` section are used.

2.8 Material parameters

The formulas used in various physical models have material specific parameters which have been fitted to the experimental data of the material. These material-specific parameter values are stored in a material config file `<mat>.cfg`, where `<mat>` is the material name.

When reading or creating a device, the material-specific parameter values are parsed from this config file and stored for every material in the device. A section of the material config file is shown below.

```
MaterName = "Silicon";
MaterType = "Semiconductor";
...
Mobility:
{
  ...
  Masetti:
  {
    mumin1 = [52.2, 44.9];
    mumin2 = [52.2, 0.];
    mu1 = [43.4, 29.];
    Pc = [0., 9.23E16];
    Cr = [9.68E16, 2.23E17];
    Cs = [3.43E20, 6.1E20];
    alpha = [0.68, 0.719];
    beta = [2., 2.];
    T0 = 300.;
  }
  ...
}

Recombination:
{
  ...
}
...
```

In the material config file, Material name is specified with the keyword `MaterName`. Type of the material is specified with the keyword `MaterType`. Following material types are supported.

- `Semiconductor`
- `Insulator`

- `Metal`

The physical models are separated into various sections, depending on their applications. In each section, the models are listed under their keywords. For example, parameters of `Masetti` model used in doping-dependent mobility calculations are listed in `Masetti` subgroup of `Mobility` group. If the parameter values are different for electrons and holes, then they are listed as a list of two floating points. For example, in the above file, `mumin1` is set to 52.2 for electrons and 44.9 for holes. Common parameters for both electrons and holes, e.g. `T0` in `Masetti` model, are listed as a single value.

2.8.1 Mole Fraction dependence

Semiconductor alloys are composed of a fraction of two elemental or compound semiconductors. For example, the alloy $\mathrm{Al}_x\mathrm{Ga}{-}1-x\mathrm{As}$ is composed of x mole fraction of AlAs and $1-x$ of GaAs. Such alloys are often used device fabrication for various purposes such as, creating quantum wells, introducing strain, etc.

Certain model parameters of these alloys are mole-fraction dependent. They are calculated by interpolating the parameter values of the end semiconductors. In the material config file of 'AlGaAs', such fraction dependent parameter values are specified as follows.

```
MaterName = "AlGaAs";
MaterType = "Semiconductor";
...
BandStructure:
{
  BandGap:
  {
    Chi0*Xdep =[0., 4.07, 0.45, 3.575, 1., 3.5];
    Eg0*Xdep = [0., 1.424, 0.45, 1.985,1.,2.17];
    alpha = 1E-4;
    beta = 270.0;
    ...
  }
  ...
```

```
}
...
Recombination:
{
  SRHRecombination:
  {
    taumaxe*Xdep =[0., 1E-6, 1., 1E-6];
    taumaxh*Xdep =[0., 1E-6, 1., 1E-6];
    taumin = [1E-6, 1E-6];
    Etrap = 0.;
    ...
  }
  ...
}
...
```

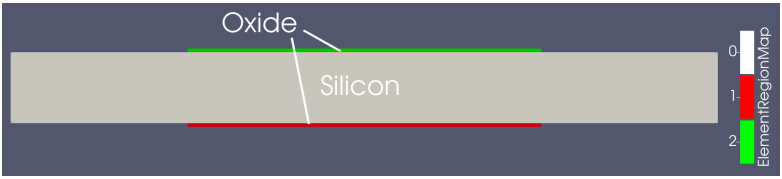
The above snippet of 'AlGaAs.par' config file shows how mole-fraction dependent values are specified for electron and hole-specific parameters and for common parameters. Mole-fraction dependent parameters are noted as `<par>*Xdep`, where `<par>` is parameter name. The parameter name is set to an `[..]` array. Each odd element of the `[..]` array specifies mole-fraction which is followed by the parameter value at that mole fraction. In the above file, band gap of AlGaAs is specified for different mole-fractions $x = 0, , 0.45, 1.0$ with the keyword `Eg0*Xdep`.

If the parameter has e,h-specific values, append 'e' to the parameter name to specify e-specific value, and append 'h' tp specify h-specific value as an `[..]` array. The array specifies mole fractions followed by the parameter value as described above. In the above file, mole-fraction dependent life-time in SRH recombination is specified for mole-fractions $x = 0, 1.0$ with the keywords `taumaxe*Xdep` and `taumaxh*Xdep`.

2.9 Running DD simulations

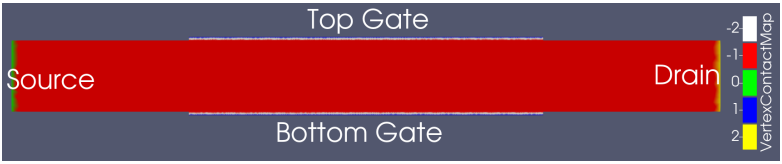
The above config file is used to perform DD simulations of a 2D FinFET structure shown in Fig. 2.1. The structure is created using the command

```
>> DDSolver str FinFET_str.cfg.
```



(a) Regions

(b) Doping

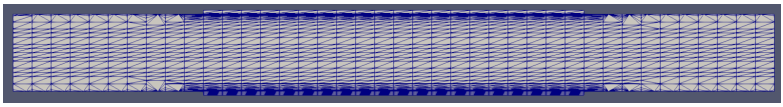


(c) Contacts

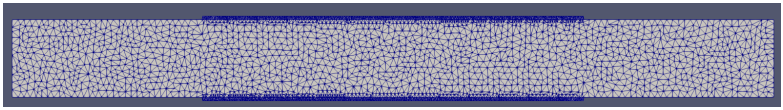
Figure 2.1: (a) 2D FinFET structure, (b) Doping concentration, and (c) Contact information set by the config file.

The above command stores structure and mesh information in the file named 'FinFET_str.h5' and also writes 'FinFET_str.xdmf' script file for visualizing that mesh in `paraview`.
`>> paraview FinFET_str.cfg`.

Once `paraview` is opened, see the `Pipeline Browser` window. Highlight `FinFET_str.xdmf` file and click the button `Apply` below in `Properties` tab. You will see the device with spatial doping concentration as shown in Fig. 2.1(b). Now, select `ElementRegionMap` instead of `DopingConcentration`. Also, open `Color Map Editor` and select the option `Interpret Values As Categories`. Elements will be colored as per the region id they belong to (see Fig. 2.1(a)).



(a) FEM mesh



(b) Triangle mesh

Figure 2.2: 2D FinFET structure generated by (a) Quadtree-based FEM mesher, and (b) *Triangle* mesher.

Similarly, select `VertexContactMap` and open `Color Map Editor` and select the option `Interpret Values As Categories`. Vertices will be colored as per the contact they belong to (see Fig. 2.1(c)). Vertices which belong to `Source` are colored green (for id 0), those of the `Gate` are blue (for id 1), while those which belong to `Drain` are colored yellow (for id 2). Vertices belonging to the semiconductor regions are red (for id -1) and insulator regions are white (for id -2).

For visualizing mesh, find the drop-down menu written `Surface` and change it to `Surface With Edges`. Mesh will be displayed as shown in Fig. 2.2(a). Change `MeshType` to `"TetMesh"` and rerun mesher. View the new mesh file with paraview. It will look like in Fig. 2.2(b). The two mesh engines available in the mesher give the same transfer characteristics on simulation.

It is not necessary to generate the structure before simulating it. The config file for generating the structure ('FinFET_str.cfg') can be specified as `Device` in `File` section of the mode solver config file 'FinFET_dev.cfg'. The solver internally generates the structure and passes it to the DD solver. The structure config file must also be present in the same folder. Once the config file is set, DD simulations can be performed using the following command

```
>> DDSolver ddsolver FinFET_dev.cfg
```

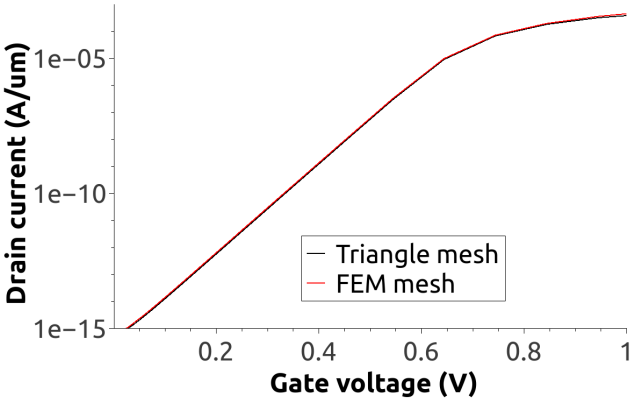


Figure 2.3: Transfer characteristics of the FinFET obtained by simulating it with the DDSolver. Transfer characteristics of the same FinFET structure created by two different meshers, namely, FEM and *Triangle* give nearly the same results.

2.10 Visualizing results

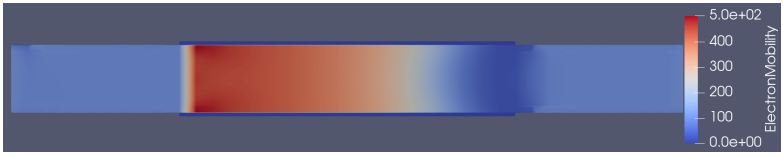
DD simulations generate xdmf files (extension *.xdmf) together with hdf5 files (extension *.h5) at the specified bias points. It also generates a csv file named 'FinFET_dev.csv' in which voltages and currents at each of the contacts at each bias point calculation are stored. Transfer characteristics stored in the 'csv' file are plotted vs. the gate voltage in Fig. 2.3.

Spatial distribution of various physical quantities at given bias points is stored in the hdf5 files. Corresponding '*dev.xdmf' files can be used to visualize the data in `paraview`.

```
>> paraview FinFETGate_3_dev.xdmf
```

Electron density, absolute electric field, abs electron current, etc. at $V_{gs} = 1.0V$ and $V_{ds} = 1.0V$ are loaded to the `paraview` using the above command and can be visualized. At $V_{gs} = 1.0V$, channel has already formed and electron density is higher at oxide-Silicon surface as seen in Fig. 2.4(a). Electron mobility is lower in the source and

(a) Electron density



(b) Electron mobility

(c) Magnitude of electron current

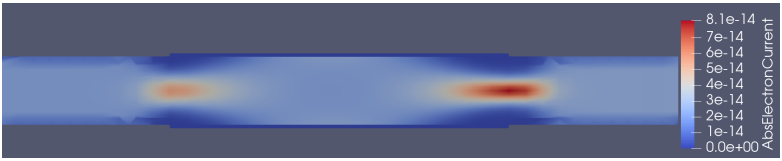
Figure 2.4: Spatial distribution of (a) electron density, (b) mobility, and (c) magnitude of electron current is plotted at the bias point of $V_{gs} = 1.0V$ and $V_{ds} = 1.0V$.

the drain due to the mobility degradation caused by heavy doping, as shown in Fig. 2.4(b). It also varies in the channel region due to the normal field dependent mobility degradation. Absolute current flows (Fig. 2.4(c)) is distributed throughout the channel region due to the narrow FinFET width which causes volume inversion as opposed to surface inversion in the bulk MOSFETs.

Fig. 2.5 plots various quantities at $V_{gs} = 50mV$ and $V_{ds} = 1.0V$. At $V_{gs} = 50mV$, the channel is off and electron density is localized at the source and the drain as shown in Fig. 2.5(a). Entire $V_{ds} = 1.0V$ drops at channel-drain junction resulting in reverse biased p-n junction. SRH generation takes place at the p-n junction as shown in Fig. 2.5(b). Small

(a) Electron density

(b) Electron mobility



(c) Magnitude of electron current

Figure 2.5: Spatial distribution of (a) electron density, (b) SRH recombination rate, and (c) magnitude of electron current is plotted at the bias point of $V_{gs} = 50mV$ and $V_{ds} = 1.0V$.

amount electrons overcome thermal barrier and cause subthreshold current flow as shown in Fig. 2.5(c).

Chapter 3

Mixed-mode Analysis

DD simulations described in Chapter 2 are performed on a single isolated FinFET device by ramping up the contact voltages. In reality, multiple such devices are connected to each other and to external components which form system of components. The DD simulator can perform coupled DD simulations on the FinFET devices coupled with the circuits using the following command:

```
>> DDSolver systemsolver FinFETSys_dev.cfg
```

In the above command, the word after `DDSolver` is the name of the program to be executed (in this case – `systemsolver`). The program name is followed by the configuration file name (in this case – `FinFETSys_dev.cfg`).

3.1 Config File

A sample configuration file of the DD Solver is provided below.

```
File: {
  Out = "FinFET";
}

Device*MOS1:
```

```
{
  File: {
    Device = "FinFET_str.cfg";
    Out = "FinFET";
  }


  Contacts: {
    gate: {Voltage = 0.0;  WorkFunction = 4.8; Type = ["Insulator"];
    source: {Voltage = 0.0; Type = ["Semiconductor"]; }
    drain: {Voltage = 0.0; Type = ["Semiconductor"]; }
  }

  Physics: {
    BandStructure: {
      CarrierDistribution: { ApproximateFermi: []; }
    }
    Mobility: {
      DopingDep: { Masetti: []; }
    }
    Recombination: {
      SRHRecombination = ["DopingDep"];
    }
  }
}

System:
{
  //SpiceCircuitFiles = [ "SpCkt1.cir", "SpCkt2.cir"];
  SpiceCircuit:
    "
    .subckt Amplifier Vc Vg Vout
    R1 Vc Vout 100.
    R2 Vs  0    10.
    R3 Vg  0    4K
    R4 Vc Vg    500.
    W1 Vout Vg Vs MOS1
    .ends
```

```
    Vsupply Vc 0 DC 0.0
    X1 Vout1 Vin Vc Amplifier
    X2 Vout Vout1 Vc Amplifier
    R1 Vout 0 1K
    .end
    "
}

Math:
{
  IterationsFC = 40;
  InnerIterationsFC = 10;
  IterationsSC = 40;
  UndampedIterations = 0;
  FCSolverTolerance = 1.;
  SCSolverTolerance = 1.;
  CircuitSolverTolerance = 1E-10;

  SolverSettings = [
            "InterpolateElecPotential",
            "InterpolateElecFermi"
            ,"InterpolateHoleFermi"
            ];
}

Plot:
{
  Quantities = ["ElectronDensity",
                "AbsElectricField",
                "ElectrostaticPotential",
                "TotalRecombination",
                "ElectronMobility",
                "AbsElectronCurrent",
                "AbsHoleCurrent"];
}

Solve:
```

```
{
  Static*Poisson: {
    Coupled: ["Poisson", "Circuit"];
    PlotTime = [0.];
  }
  Static*Poisson2: {
    Coupled: ["Poisson", "Electron", "Circuit"];
    PlotTime = [0.];
  }

  Quasistationary*SupplyVRamp:
  {
    initstep = 1E-3; minstep = 1E-5; maxstep = 0.1; incr = 1.35; decr
    Ramp: {
      Voltage*Vsupply = 1.0;
    }
    Coupled: ["Poisson", "Electron", "Circuit"]
    PlotTime = [0., 0.05, 0.5, 1.0];
  }

  Quasistationary*InputVRamp:
  {
    initstep = 1E-3; minstep = 1E-5; maxstep = 0.1; incr = 1.35; decr
    Ramp: {
      Voltage*Vinput = 1.0;
    }
    Coupled: ["Poisson", "Electron", "Circuit"]
    PlotTime = [0.,0.05, 0.5, 1.0];
  }
}
```

The above config file can be used for simulating a system of two inverter circuits made of FinFET devices. It has a similar composition as that of the config file used for simulating single device in Chapter 2. Additional keywords in the file and their functionality is described below.

3.2 File section

Comparing the above config file with that in Chapter 2, one may notice two file sections. The first file section belongs to the 'system' of devices. Prefix of the output files of mixed-mode simulation is provided as an argument `out`.

Notice another `File` subsection inside `Device` section in the above file. It is specific to the device. Functionality of the `Device` section is described below.

3.3 Device section

Notice a `Device` section named 'Mos1' in the above file. This section specifies the device structure file (in `File` subsection), contact settings (in `Contacts` subsection), and active physical models in the device (in `Physics` subsection). Meaning of these subsections is the same as described in Chapter 2. This `Device` section defines a FinFET named 'MOS1'. This device may be referenced in the circuit file (or circuit text) in `System` section by the keyword `MOS1`. In the circuit file, the device instantiated in the same way as a sub-circuit with the same number of output pins as the number of contacts of the device.

3.4 System section

The system section describes how various components of the circuit, such as resistors, capacitors, or inductors together with the device (here 'MOS1') are connected to each other in the circuit. This connectivity is set by a 'Spice' script provided with the argument `SpiceCircuit`, as shown in the `System` section above. Or, it can be specified in circuit file names ('*.cir'), specified with the argument `SpiceCircuitFiles` as a list of comma separated files.

3.5 Math section

Global math settings for the system solver are set in `Math` section of the file. They have the same meaning as described in Sec 2.5. An

additional argument `CircuitSolverTolerance` specifies the dividing factor for the circuit solver residual.

3.6 Plot section

The argument `Quantities` specifies all the physical quantities stored in the hdf5 file, similar to Sec 2.6.

3.7 Solve section

Solve section of the config file specifies which voltage/current sources to be ramped to the specified value. Most of the arguments specified in `Solve` section of the system simulation config file are similar to those described in Sec 2.6. The additional arguments are described here.

The solve section can list one or many `Static`, `Quasistationary`, or `Transient` ramps, which have the same meanings as described in Sec 2.6. In system solve section, the coupled argument has an additional keyword – `Circuit`. It specifies that the `Circuit` is coupled with other equations, such as `Poisson`, `Electron`, or `Hole` continuity equations.

Also, notice that the `Ramp` subsection in the system config file *does not* list the contact names to be ramped. Instead, it specifies voltage to which the voltage source `<vname>` to be ramped, with the keyword `Voltage*<vname>`.

3.8 Running mixed-mode simulations

The above config file is used to perform mixed-mode simulations of an amplifier sub-circuit specified by the name `Amplifier` in the `SpiceCircuit` text in `System` section. The DD system device is named `MOS1` and is represented by the same name in the spice circuit file as shown in the line below.

```
W1 Vout Vg Vs MOS1
```

The device instance starting with the letter `W` is a DD system device instance. It is followed by the net names connected to the contacts

of the DD device. It is followed by the device name `MOS1`. The DD device is a FinFET whose structure shown in Fig. 2.1. The structure is created using the command

```
>> DDSolver str FinFET_str.cfg.
```

Two `Amplifier` sub-circuit instances `X1` and `X2` are instantiated in `SpiceCircuit`. These amplifier instances are connected back-to-back in the main circuit. Thus, two instances of `MOS1` are simulated in mixed-mode simulation. The mixed-mode system simulation is performed using the following command.

```
>> DDSolver systemsolver FinFETSys_dev.cfg.
```

Spatial distributions of various quantities at specific bias points are stored in hdf files named '<out>_<dev>_<instname>_<deviceid><ramp>_<fileid>_dd.h5', where <out> is the prefix specified by `Out`, <dev> is device name (here MOS1), <instname> is name of the circuit instance (here 'W1'), <deviceid> is a unique device id in the system, <ramp> is ramp name, and <fileid> is bias point id. In this case, the files are saved with the name 'FinFET_MOS1_W1_0GateRamp_0_dd.h5'. The file can be opened using the following command.

```
>> paraview FinFET_MOS1_W1_0GateRamp_0_dd.xdmf
```

Currents and voltages at all the bias points are saved in a 'csv' file named '<out>_<deviceid>_dev.csv'. They can be visualized using any plotting tool, for example `qtiplot`.

Chapter 4

Small-signal Analysis

DD simulations can be performed on a single device to calculate small-signal AC response of the device at a specific Direct Current (DC) bias. This is done by first ramping up the contact voltages quasi-stationarily to the given DC bias point. Then, Jacobian at the DC bias is used to calculate AC response of the device.

```
>> DDSolver ddsolver FinFET_dev.cfg
```

4.1 AC Analysis Config File

A configuration file performs AC analysis when an `ACAnalysis` section is presented.

```
File: {
  Device = "FinFET_str.cfg";
  Out = "FinFET";
}

Contacts: {
  gate: {Voltage = 0.0;  WorkFunction = 4.8; Type = ["Insulator"]; }
  source: {Voltage = 0.0; Type = ["Semiconductor"]; }
  drain: {Voltage = 0.0; Type = ["Semiconductor"]; }
}
```

```
Physics: {
  BandStructure: {
    CarrierDistribution: { ApproximateFermi: []; }
  }
  Mobility: {
    DopingDep: { Masetti: []; }
  }
  Recombination: {
    SRHRecombination = ["DopingDep"];
  }
}

Math:
{
  IterationsFC = 40;
  InnerIterationsFC = 10;
  IterationsSC = 40;
  UndampedIterations = 0;
  FCSolverTolerance = 1.;
  SCSolverTolerance = 1.;

  SolverSettings = [
            "InterpolateElecPotential",
            "InterpolateElecFermi"
            ,"InterpolateHoleFermi"
            ];
}

Plot:
{
  Quantities = ["ElectronDensity",
    "AbsElectricField",
    "ElectrostaticPotential",
    "TotalRecombination",
    "AbsElectronCurrent"];
  ACQuantities = ["ElectronDensity",
    "AbsElectricField",
```

```
    "ElectrostaticPotential"];
}

Solve:
{
  Static*Poisson: {
    Coupled: ["Poisson"];
    Plot: { Time = [0.]; }
  }

  Quasistationary*Drain: {
    initstep = 1E-3; minstep = 1E-5; maxstep = 0.1;
    incr = 1.35; decr = 2;
    Ramp: {
      Voltage*drain = 1.;
    }
    Coupled: ["Poisson","Electron"]
    PlotTime = [0., 0.5, 1.0];
  }

  Quasistationary*Gate: {
    initstep = 1E-3; minstep = 1E-5; maxstep = 0.1;
    incr = 1.35; decr = 2;
    Ramp: {
      Voltage*gate = 1.0;
    }
    Coupled: ["Poisson","Electron"]
    PlotTime = [0., 0.05, 0.5, 1.0];
    ACAnalysis: {
      Nodes = ["gate", "source", "drain"];
      Time = [0.99];
      MinFrequency = 1E2; MaxFrequency = 1E6;
      PointsPerDecade = 3;
      PlotFrequency = [1E3, 1E6];
    }
  }
}
```

All the sections in the above config file have the same meaning as described in Chapter 2. An addition subsection called `ACAnalysis` is added in the last `Quasistationary` section 'Gate'. This command is described below.

4.2 ACAnalysis section

In small-signal analysis, admittance between every permutation of the contact names listed in `Nodes` list is calculated at each frequency in the frequency list. The frequency list is created by adding frequency points from start frequency given by the argument `MinFrequency` till end frequency specified by `MaxFrequency`. Number of points per decade of frequency is set by the argument `PointsPerDecade`. For calculating admittance, a unit amplitude AC voltage source is connected to each of the contacts while setting other contacts to the AC ground. Phasor currents from all the contacts are calculated for each of the configurations, which give the admittances and the capacitances between each pair of the contacts. This procedure is repeated on each of the contacts by connecting AC voltage source to it and measuring currents from all other contacts.

In the given example, the admittances (A) and the capacitances (C) shown in the following matrix are calculated.

$$\begin{bmatrix} \partial I_\mathrm{s} \\ \partial I_\mathrm{g} \\ \partial I_\mathrm{d} \end{bmatrix} = \begin{bmatrix} A_\mathrm{ss} & A_\mathrm{gs} & A_\mathrm{ds} \\ A_\mathrm{sg} & A_\mathrm{gg} & A_\mathrm{dg} \\ A_\mathrm{sd} & A_\mathrm{gd} & A_\mathrm{dd} \end{bmatrix} \cdot \begin{bmatrix} \partial V_\mathrm{s} \\ \partial V_\mathrm{g} \\ \partial V_\mathrm{d} \end{bmatrix} + i\omega \cdot \begin{bmatrix} C_\mathrm{ss} & C_\mathrm{gs} & C_\mathrm{ds} \\ C_\mathrm{sg} & C_\mathrm{gg} & C_\mathrm{dg} \\ C_\mathrm{sd} & C_\mathrm{gd} & C_\mathrm{dd} \end{bmatrix} \cdot \begin{bmatrix} \partial V_\mathrm{s} \\ \partial V_\mathrm{g} \\ \partial V_\mathrm{d} \end{bmatrix} \tag{4.1}$$

4.3 Running small-signal analysis

The above config file is used to perform DD simulations of a 2D FinFET structure shown in Fig. 2.1. The structure is created using the command

```
>> DDSolver str FinFET_str.cfg.
```

The above command stores structure and mesh information in the file named 'FinFET_str.h5' and also writes 'FinFET_str.xdmf' script file for visualizing that mesh in `paraview`.

```
>> paraview FinFET_str.xdmf.
```

AC analysis is performed at the bias point at the fictional time $t > 0.99$, i.e. at $V_{gs} = 1.0V$ and $V_{ds} = 1.0V$. The device config file can be run by the following command.

```
>> DDSolver ddsolver FinFET_dev.cfg.
```

The simulator performs `Quasistationary` simulations till the required bias point is reached. Jacobian at that bias point is used to perform small-signal analysis on the device as explained above. Admittances and capacitances are written in a csv file named '<out>_<rampname>_<biasid>_ac.csv'. The csv file can be imported in any of the plotting tools such as `qtiplot` and frequency dependent capacitances and admittances can be plotted. Opening the csv file in a text editor will show the following lines,

```
0->Source
1->Gate
2->Drain
Frequency, A(0 0), C(0 0), A(0 1), C(0 1), A(0 2), C(0 2),...
```

First lines of the file show mapping of the contacts with their ids. The line starting with `Frequency` lists the headers of the data. First column corresponds to `Frequency` in Hz followed by admittance at contact `0` (here, `Source`) due to the small-signal voltage ∂V_s is applied at the same contact `0`. This is represented by `A(0 0)`. Next column corresponds to the capacitance `C(0 0)`. The column after that corresponds to `A(0 1)`: the admittance between the `Source` and the `Gate`. All the column headers are named in the same fashion.

Applying small-signal voltage ∂V_c at contact c results in small variations in the spatial distribution of electron/hole density/Fermi levels, electrostatic potential, etc. These small changes can have both real and imaginary components (Note, that ∂V_c is always applied

with zero imaginary component). Real and imaginary components of the spatial distributions of these quantities are stored at specific frequencies given by `PlotFrequency`. They are stored in hdf files named '<out> _ <rampname>_ACFreqId_<id>_<cont>_ac.h5'. Corresponding 'xdmf' files are also stored. They can be visualized in `paraview` as follows.

```
>> paraview FinFET_Gate_ACFreqId_1_gate_dev.xdmf
```

Spatial distribution of real part of small-signal variations in electron density arising from small-signal voltages ($f = 1$kHz) at the `Source`, `Gate`, and `Drain` are shown in Fig. 4.1. Comparing the effect of AC voltage at the source (Fig. 4.1(a)) and at the drain (Fig. 4.1(c)) shows, that small variations in ∂V_{d} are confined to the drain. Due to the channel pinch off at the channel-drain junction, the drain 'electron well' is secluded from the rest of the device resulting in such an effect.

(a) Effect of small-signal voltage at source

(b) Effect of small-signal voltage at gate

(c) Effect of small-signal voltage at drain

Figure 4.1: Spatial small-signal variation of electron density because of application of small-signal voltage at (a) source, (b) gate, and (c) drain.

Chapter 5

Drift-diffusion Solver Equations

Spatial distribution of free and fixed charges in semiconductor and insulator regions of the devices satisfy *Poisson equation* together with the boundary conditions at any given time. Similarly, electron and hole concentration in the semiconductor regions must also satisfy the carrier continuity equations together with the boundary conditions. The DD simulator solves these three equations, Poisson equation, electron continuity, and hole continuity equations together to determine electrostatic potential, electron, and hole concentrations. These equations are described below.

5.1 Poisson equation

Poisson equation in the presence of free charges is given by,

$$-\nabla^2\Psi(\vec{r}) = \rho_C(\vec{r}) + n(\Psi(\vec{r})) - p(\Psi(\vec{r})) \tag{5.1}$$

where $\rho_C(\vec{r})$ is net concentration of fixed charges which includes dopants, bulk, and interface charges, and charged traps, p is the hole concentration, and n is the electron concentration. $\Psi(\vec{r})$ is the spatially varying electrostatic potential in the device. p and n show

strong non-linear dependence on the local electrostatic potential $\Psi(\vec{r})$. Therefore, Eq. 5.1 is a non-linear equation of $\Psi(\vec{r})$. In DD simulator, damped Newton's method is used to solve this equation.

Electron ($n(\vec{r})$) and hole ($p(\vec{r})$) concentrations are calculated from the Density of States (DOS) and the carrier Fermi energy level by using one of the 'Boltzmann' equation, 'Approximate Fermi' equation, or Fermi equation. The equations and underlying approximations are described in detail in the next chapters.

Fixed charges ($\rho_C(\vec{r})$) are created at any location in the bulk material of the semiconductor by ionization of dopant atoms, charged traps, and fixed charges originating from material defects.

The DD simulator reads net dopant concentration ($\rho_D(\vec{r})$) from the device structure and adds it to the total fixed charges at each location. Thus, *complete ionization* is assumed in DD calculations.

Fixed charges originating from material defects are added to the $\rho_C(\vec{r})$. Concentration of charged traps is calculated using the principle of detailed balance and added to the $\rho_C(\vec{r})$.

Fixed charges and traps can also exist at the material interfaces. The simulator calculates 'sheet concentration' of these charges, multiplies with the interface area and then adds to the total charge concentration.

5.2 Carrier continuity equation

Carrier continuity equation implies, that under quasi-stationary conditions, divergence of carrier current densities ($\vec{J_n}(\vec{r})$ and $\vec{J_p}(\vec{r})$) at any spatial location equals carrier generation rate at that location. This is given by,

$$\frac{\partial n}{\partial t} = \nabla \cdot \vec{J_n}/q - (R_n - G_n) \tag{5.2a}$$

$$\frac{\partial p}{\partial t} = -\nabla \cdot \vec{J_p}/q - (R_p - G_p) \tag{5.2b}$$

$$\tag{5.2c}$$

where, are electron and hole currents. $R_{n/p}$ and $G_{n/p}$ are recombination and generation rates of electrons/holes. Under quasi-stationary conditions $\frac{\partial n}{\partial t}$ and $\frac{\partial p}{\partial t}$ are both zero. Note, that $\vec{J}_{n/p}$ as well as

$R_{n/p} - G_{n/p}$ are nonlinear functions of Ψ, p, and n. Therefore, Eq. 5.2 form a system of nonlinear equations of the solution variables. In DD simulator, damped Newton's method is used to solve these equations.

In the case of transient ramps, the above system of equations must be evolved using one of the time-stepping algorithms. In DD simulator, Backward Euler (BE) method is implemented to solve time evolution solutions of time-dependent Partial Differential Equation (PDE).

At any location $\vec{r}$, electron and hole current ($\vec{J_n}(\vec{r})$ and $\vec{J_p}(\vec{r})$) arise from drift of the carriers in the electric field ($\vec{E} = \nabla\Psi$) and diffusion of the carriers due to the concentration gradient (∇n and ∇p).

$$\vec{J_n} = n \cdot q \cdot \mu_n \cdot \nabla\Psi - D_n \nabla n \tag{5.3a}$$

$$\vec{J_p} = p \cdot q \cdot \mu_p \cdot \nabla\Psi + D_p \nabla p \tag{5.3b}$$

Here, Ψ is the electrostatic potential, D_n and D_p are electron and hole diffusivities, μ_n and μ_p are mobilities, and q is electronic charge. Using Einstein relation for carrier diffusivities ($D_{n/p} \approx \mu_{n/p} \cdot \frac{kT}{q}$) reduces the above equation to,

$$\vec{J_n} = -n \cdot q \cdot \mu_n \cdot \nabla\Phi_n \tag{5.4a}$$

$$\vec{J_p} = -p \cdot q \cdot \mu_p \cdot \nabla\Phi_p \tag{5.4b}$$

Here, $\Phi_{n/p}$ are electron and hole Fermi energies. In DD simulator, the carrier current densities obtained from the Einstein approximations given by Eq. 5.4 are used in continuity Eq. 5.2. The continuity equations are solved to obtain carrier concentrations and carrier Fermi energies.

5.3 Coupled and self-consistent solver

In DD simulator, the above three equations – Poisson equation (Eq. 5.1), electron continuity (Eq. 5.2a), and hole continuity (Eq. 5.2b) – are solved simultaneously. All these equations are nonlinear in solution variables Ψ, p, and n. Thus, solving continuity equation changes electron and hole concentrations which necessitates solving Poisson equation again. In a self-consistent solver, these equations are solved

one after the other repeatedly till convergence is reached (i.e. solutions change little in subsequent iterations). Specifying the keywords `SelfConsistent: ["Poisson", "Electron"]` in the ramp commands in `Solve` section sets a self-consistent solver for Ψ and n.

Alternately, all the three equations can be solved as coupled equations. A coupled solver is also provided in DD simulator. Specifying a `Coupled` keyword in the ramp command in `Solve` section `Coupled: ["Poisson", "Electron"]` sets a coupled solver for Ψ and n.

5.4 Domain Boundary Conditions

5.4.1 Mirror BC

Simulation of drift-diffusion equations within a finite simulation domain activate *zero flux* BC at the domain boundary, by default. For *Poisson* equation, they are given by Eq. 5.5.

$$\nabla\Psi(\vec{r}) \cdot \hat{n} = \vec{F}(\vec{r}) \cdot \hat{n} = 0 \, \forall \vec{r} \in \delta\Omega \tag{5.5}$$

Similarly, for *electron* continuity equation, mirror BCs are given by Eq. 5.6.

$$\nabla\Phi_n(\vec{r}) \cdot \hat{n} = \vec{J_n}(\vec{r}) \cdot \hat{n} = 0 \, \forall \vec{r} \in \delta\Omega \tag{5.6}$$

Also, for *hole* continuity equation, mirror BCs are given by Eq. 5.7.

$$\nabla\Phi_p(\vec{r}) \cdot \hat{n} = \vec{J_p}(\vec{r}) \cdot \hat{n} = 0 \, \forall \vec{r} \in \delta\Omega \tag{5.7}$$

Here, $\hat{n}$ is unit normal at $\vec{r}$ along the domain boundary ($\delta\Omega$).

When the simulation domain is continuous at the boundary, the above BCs are equivalent to *mirror* BCs. Therefore, it is advised to set the boundary of the simulation domain such that the device is mirrored on the other side of the domain.

5.4.2 PBC

PBCs can be set at the extrema along `X` and `Y` axes in 2D device, and `X`, `Y` and `Z` axes in 3D devices. Two types of periodic BCs are available - *Mortar* and *Robin* PBC.

Mortar PBC

In 'mortar' PBC, vertices at the extrema along the axes of PBC are *stitched* together. If PBC is specified at `Xmin`, vertices along the domain boundary at minimum of `X` are stitched to those at the maximum of `X`. Alternately, if PBC is specified at `Xmax`, vertices along the domain boundary at maximum of `X` are stitched to those at the minimum of `X`. Thus, for each of the equations, PBCs are given by,

$$\begin{aligned} \forall \vec{r}_m \in \Omega(x_{\min}) \,\text{and}\, \vec{r}_M \in \Omega(x_{\max}) & \\ \Psi(\vec{r}_m) = \Psi(\vec{r}_M) & \quad (5.8) \\ \Phi_n(\vec{r}_m) = \Phi_n(\vec{r}_M) & \quad (5.9) \\ \Phi_p(\vec{r}_m) = \Phi_p(\vec{r}_M) & \quad (5.10) \\ & \quad (5.11) \end{aligned}$$

Same procedure is applied for all vertices at the boundaries of `Y` and (in 3D) for`Z` axes.

Robin PBC

In 'robin' PBC, vertices at the extrema along the axes of PBC are *coupled* together. If PBC is specified at `Xmin`, vertices along the domain boundary at minimum of `X` are coupled to those at the maximum of `X`. Alternately, if PBC is specified at `Xmax`, vertices along the domain boundary at maximum of `X` are coupled to those at the minimum of `X`. Thus, PBCs for each of the equations are given by,

$$\begin{aligned} \forall \vec{r}_m \in \Omega(x_{\min}) \,\text{and}\, \vec{r}_M \in & \Omega(x_{\max}) \\ \alpha \cdot (\Psi(\vec{r}_m) - \Psi(\vec{r}_M)) - \vec{F}(\vec{r}_m) = 0 & \quad (5.12) \\ \alpha \cdot (\Phi_n(\vec{r}_m) - \Phi_n(\vec{r}_M)) - \vec{J}_n(\vec{r}_m) = 0 & \quad (5.13) \\ \alpha \cdot (\Phi_p(\vec{r}_m) - \Phi_p(\vec{r}_M)) - \vec{J}_n(\vec{r}_m) = 0 & \quad (5.14) \\ & \quad (5.15) \end{aligned}$$

Here, α is an adjustable parameters. It can be set in device `Math` section by the keyword `RobinBCFactor`. It is set to 1 by default.

5.4.3 Defining PBC

Following flag must be set in device `Math` section to activate periodic BCs -

`(Robin|Morter)PBCAt(X|Y|Z)(min|max)`

Following examples show usage of the flag.

- `RobinPBCAtXmin` : Robin PBC along `X` axis at minimum x.
- `RobinPBCAtYmax` : Robin PBC along `Y` axis at maximum y.
- `MorterPBCAtXmax` : Mortar PBC along `X` axis at maximum x.

By default, each vertex at `Xmin` boundary is paired with the *nearest* vertex at `Xmax` boundary. If the two vertices have identical y and z coordinates, then this construction is accurate. Otherwise, it introduces unphysical skewing of the device geometry at the boundary. In order to avoid it, `PBCUseVertexInterp` flag is set in `Math` section. It binds the solutions (Psi, Phi_n, and Phi_p) at the vertex at `Xmin` to the linearly interpolated values of the respective solutions at `Xmax`. This avoids unphysical skewing.

5.5 Electrical Boundary conditions

Boundary conditions of the above equations depend on what type of electrical BC has been set at the device mesh boundary region.

5.5.1 Ohmic contacts

Semiconductor contacts are, by default, ohmic contacts. If they are connected to an external circuit (e.h. mixed-mode simulations), then the contact resistance of $10^{-3}\Omega$ is introduced by default. In case of the contact voltage ramp, no contact resistance is introduced. Charge neutrality is always assumed at Ohmic contacts.

$$n_0 - p_0 = N_{D,net} \tag{5.16a}$$

$$n_0 \times p_0 = n_i^2 \tag{5.16b}$$

where n_0 and p_0 are equilibrium electron and hole concentrations, n_i is the intrinsic density, and $N_{D,net}$ is net doping concentration. These conditions can be evaluated analytically for Boltzmann and approximate Fermi statistics. These conditions allow us to specify electrostatic potential (Ψ) and contact Fermi energy ($\phi_{F,e/h}$) as a function of doping and contact voltage. Contact electrostatic potential is calculated for a given contact voltage V_C as follows,

$$\psi_C = V_C + \frac{kT}{q}\mathrm{a}sinh\left(\frac{N_{D,net}}{2n_i}\right) \tag{5.17a}$$

$$\phi_{F,e/h} = V_C \tag{5.17b}$$

This boundary condition is used at Ohmic contacts while solving Poisson and continuity equations.

If `eRecVelocity` (v_e in cm/sec) and `hRecVelocity` (v_h in cm/sec) are specified in the contact definition in `Electrode` section, contact Fermi is not determined by Eq. 5.17b. Instead the following BCs are imposed at the contact.

$$\vec{J}_n \cdot \hat{n} = qv_e(n - n_0) \tag{5.18a}$$

$$\vec{J}_p \cdot \hat{n} = -qv_h(p - p_0) \tag{5.18b}$$

The above equations are solved together with carrier continuity equation to determine contact Fermi energy.

5.5.2 Schottky contacts

A Semiconductor contact is specified as a Schottky contact by adding an argument `SchottkyBarrier` together with Schottky barrier value (Ψ_B) at the contact. At the Schottky contact, the following BCs imposed.

$$\psi_C = V_C - \Psi_B + \frac{kT}{q}\ln\left(\frac{N_C}{n_i}\right) \tag{5.19a}$$

$$\vec{J}_n \cdot \hat{n} = qv_e(n - n_B) \tag{5.19b}$$

$$\vec{J}_p \cdot \hat{n} = -qv_h(p - p_B) \tag{5.19c}$$

$$n_B = N_C \exp\left(\frac{-q\Psi_B}{kT}\right) \tag{5.19d}$$

$$p_B = N_V \exp\left(\frac{-E_g + q\Psi_B}{kT}\right) \tag{5.19e}$$

where V_C is the contact voltage, v_e and v_h are e and h thermionic emission velocities, N_C and N_V CB and VB DOS. Thermionic emission velocities v_e and v_h are set to, respectively, 2.5×10^6cm/sec and 1.9×10^6cm/sec, by default. They can be modified by specifying `eRecVelocity` and `hRecVelocity` in the contact definition in `Electrode` section.

5.5.3 Contact Resistances

When a contact is connected to an external circuit, a contact resistance of $10^{-3}\Omega$ is introduced, by default. Alternately, a contact resistance (in Ohms) may be specified in the contact definition using the argument `Resistance`. Such resistive contact may be viewed as having an external node and an internal node. Contact resistance is connected between the two. User specified contact voltage is applied at the external node. Voltage on the internal node is determined by solving the equation,

$$\frac{V_C - \phi_C}{R_C} = I_C(\Phi_C) \tag{5.20}$$

Note, that total current passing through the contact I_C depends on voltage on the internal node (ϕ_C). The above equation is coupled with the continuity equations and solved to determined ϕ_C.

5.6 Nonlinear solver

DD simulator solves Poisson equation (Eq. 5.1), electron, and hole continuity equations (Eq. 5.2a and Eq. 5.2a) to obtain electrostatic potential, and e and h densities. Since the equations are nonlinear in solution variables, they need to be solved using a nonlinear solver, such as Newton's solver. The DD simulator uses a damped Newton's solver reported in Bank et al.

In this solver, a nonlinear system of equations $\vec{f}(\vec{x}) = 0$ is solved by Newton's method. It involves calculating *Jacobian* matrix ($\mathbf{g}'(\vec{x}_i)$)

at a current step i, and solve the following equation to get solution update ($\vec{dx}_i$) for the next step, $i+1$.

$$\mathbf{g}'(\vec{x}_i) \cdot \vec{dx}_i = -g(\vec{x}_i) \tag{5.21a}$$

$$\vec{x}_{i+1} = \vec{x}_i + \gamma \vec{dx}_i \tag{5.21b}$$

Here, γ is the damping parameter calculated such that $\frac{\|g(\vec{x}_{i+1})\|}{\|g(\vec{x}_i)\|} < 1$ and $\frac{\|g(\vec{x}_{i+1})\|}{\|g(\vec{x}_i)\|}$ is as close to unity as possible.

5.6.1 Available Linear Solvers

Linear matrix equation (Eq. 5.21a) in the above described Newton's method can solved by any of the following matrix solvers.

- `Pardiso` : This solver is provided with Intel math kernel libraries (MKL). It can offer simulation speed-up on Intel processors compared to the other solvers. It is used by default, when the DD solver version *with* mkl acceleration is installed. Note, that `Pardiso` solver cannot be selected in the DD solver without mkl acceleration.

- `SuperLU` : This solver is provided with *SuperLU* package. It is the default solver, when the DD solver *without* mkl acceleration is installed. It can be activated by adding `SuperLUSolver` flag to the `Settings` in `Math` section of the config file.

- `ILS` : An iterative linear solver based on *GMRES* method can be selected. It uses *Incomplete LUT* preconditioner. It can be activated by adding `ILSMethod` flag to the `Settings` in `Math` section of the config file.

If the iterative solver is selected by specifying `ILSMethod` flag, then the following parameters can be used to configure 'GMRES' method together with the preconditioner based on 'incomplete LU'.

- `ILSMaxIter` : It sets maximum iterations performed in GMRES method.

- `ILSRestartIter` : It specifies iterations after which GMRES reset is performed.

- `ILSGMRESTol` : It sets solver tolerance in GMRES method.
- `ILSPrecondDropTol` : It specifies 'drop-tolerance' of the ILUT method. If the matrix entry is less than the tolerance value then it is ignored.
- `ILSPrecondDropTol` : It specifies fill-factor in the incomplete LU method.

5.6.2 Convergence criterion

Nonlinear iterations as described in the Damped-Newton algorithm continue until convergence criterion is reached. By default, when the norm of the residual $\|g(\vec{x}_i)\| < 1.0$, the Newton's iterations exit. If `RelativeError` is specified in `SolverSettings` in `Math` section, then relative error criterion is used for convergence check. At each step, relative error is calculated by subtracting previous solution from the current solution and dividing it by the current solution. It is represented as follows.

$$\frac{1}{\epsilon_R}\frac{1}{N}\sum_{eq,n}\frac{|x(eq,n,i)-x(eq,n,i-1)|}{|x(eq,n,i)|+\epsilon_{ref}(eq)} < 1 \tag{5.22}$$

Here, N is the number of nodes, eq is the equation (Poisson, e or h continuity), i is Newton's step. ϵ_R is the relative error set by argument `RelativeError`, $\epsilon_{ref}(eq)$ is equation-wise relative error criterion, set by `RelativeErrorElectron`, `RelativeErrorHole`, and `RelativeErrorPoisson` for e, h continuity, and Poisson equation, respectively.

Chapter 6

Band-structure models

Physical quantities related to band-structure of the semiconductor material are described in this chapter. Their parameterization in the material config file of the DD simulator is also explained. Note, that all the models and their parameters are listed in `BandStructure` part of the material file.

6.1 Band-gap

Energetic separation of lowest energy state of the CB and the highest energy state of the VB is the band gap of the semiconductor. It is set in `BandGap` section by the argument `Eg0` together with the temperature `T0` at which the band gap has been measured.

Temperature dependence of the band gap is modeled by Varshney's relation as follows.

$$E_g(T) = E_g(T_0) + \frac{\alpha T_0^2}{T_0 + \beta} - \frac{\alpha T^2}{T + \beta} - E_{bgn} \tag{6.1}$$

Here, T is the lattice temperature, and T_0 is the temperature at which $Eg(T_0)$ has been measured. Also, α and β are Varshney parameters set in `BandGap` section by the arguments `alpha` and `beta` respectively. E_{bgn} is the band gap narrowing in heavily doped semiconductor.

6.1.1 Band-gap Narrowing

Heavy doping in the semiconductor results in reduction of the band gap. This band-gap narrowing (E_{bgn}) depends on the doping levels in the semiconductor. Various models have been reported in the literature to model this effect. They have been listed below along with the fitting parameters.

Bennett-Wilson model

In `BennetWilson` model, the band-gap narrowing is modeled as follows.

$$E_{bgn} = \begin{cases} E_{ref} \left[\ln \left(\frac{N_{tot}}{N_{ref}} \right) \right]^2 & if\ N_{tot} \geq N_{ref} \\ 0 & otherwise \end{cases} \tag{6.2}$$

The parameters, E_{ref} and N_{ref} are listed under `BennetWilson` section in `BandStructure` part of the material file.

delAlamo model

In `DelAlamo` model, the band-gap narrowing is modeled as follows.

$$E_{bgn} = \begin{cases} E_{ref} \cdot \ln \left(\frac{N_{tot}}{N_{ref}} \right) & if\ N_{tot} \geq N_{ref} \\ 0 & otherwise \end{cases} \tag{6.3}$$

The parameters, E_{ref} and N_{ref} are listed under `DelAlamo` section in `BandStructure` part of the material file.

Slotboom model

In `Slotboom` model, the band-gap narrowing is modeled as follows.

$$E_{bgn} = E_{ref} \cdot \left[\ln \left(\frac{N_{tot}}{N_{ref}} \right) + \sqrt{\left(\ln \left(\frac{N_{tot}}{N_{ref}} \right) \right)^2 + 0.5} \right] \tag{6.4}$$

The parameters, E_{ref} and N_{ref} are listed under `Slotboom` section in `BandStructure` part of the material file.

6.1.2 Electron affinity

Energetic separation between the CB edge and the vacuum energy level is called electron affinity χ of the material. Its temperature dependence is modeled by the following expression.

$$\chi(T) = \chi_0 + \frac{(\alpha + \alpha_2)T^2}{2(T + \beta + \beta_2)} + \gamma_{bgn} \cdot E_{bgn} \tag{6.5}$$

Here, χ_0 is set by `Chi0`, α, β are Varshney parameters, and α_2, β_2 are additional parameters to fit temperature dependence of $\chi(T)$. The parameters α_2, β_2 are set by the arguments `alpha2` and `beta2` in `BandGap` section.

Band-gap-narrowing lowers CB energy and increases VB energy. γ_{bgn} is a fraction of band-gap-narrowing by which CB energy is lowered. It is set by `BgnToChi` argument in `BandGap` section.

6.2 Carrier distribution

Energetic distribution of electrons and holes follows Fermi-Dirac formula as follows.

$$n(\vec{r}) = N_C \mathcal{F}_{1/2}\left(\frac{q(\phi_n - E_C)}{kT}\right) \tag{6.6a}$$

$$p(\vec{r}) = N_V \mathcal{F}_{1/2}\left(\frac{q(E_V - \phi_p)}{kT}\right) \tag{6.6b}$$

where N_C and N_V are the CB and VB DOS values, E_C and E_V are local CB edge and VB edge energies. $\mathcal{F}_{1/2}(\eta)$ is Fermi-Dirac integral given below.

$$\mathcal{F}_{1/2}(\eta) = \int_{\epsilon=0}^{\infty} \frac{\epsilon^{1/2}}{1 + \exp(\epsilon - \eta)} d\epsilon \tag{6.7}$$

The above integral has no closed form solution. In DD simulations, it is often required to calculate carrier density from Fermi energy and vice versa. A closed form solution of the above integral would highly speed up the computation. For that purpose, the above integral is typically approximated to analytic expressions. The following two approximations can be selected for DD simulations.

6.2.1 Boltzmann approximation

By default, Boltzmann approximation is applied in DD simulations. In Boltzmann approximation, Fermi-Dirac integral is analytically expressed as follows,

$$\mathcal{F}_{1/2}(\eta) = \exp \eta \tag{6.8}$$

Boltzmann distribution is fairly accurate if the carrier Fermi energy $\phi_{n/p}$ lies in the band-gap of the semiconductor. Accuracy of this approximation is low in the heavily doped areas of the device. In practice, the above approximation gives a good description of device characteristics with low computational burden.

6.2.2 Approximate Fermi

In approximate Fermi approximation, electron and hole density is calculated as follows.

$$\mathcal{F}_{1/2}(\eta) = \frac{1}{\exp -\eta + 0.27} \tag{6.9}$$

Approximate Fermi distribution can be selected by specifying `ApproximateFermi` argument in `CarrierDistribution` part of 'global' `Physics` section as follows.

```
Physics: {
BandStructure: {
CarrierDistribution: { ApproximateFermi: []; }
}
}
```

Approximate Fermi distribution can only be activated in the global physics section. It cannot be set in region/material physics section.

6.2.3 Density of States

The DOS factor appearing in Eq. 6.6 is given by,

$$N_{C/V} = 2 \left(\frac{2\pi m_{C/V}^{dos} k_B T}{\hbar^2} \right)^{3/2} \tag{6.10}$$

whereas, intrinsic carrier concentration is given by

$$n_i^2 = N_C \cdot N_V \cdot \exp\left(\frac{-qE_g}{kT}\right) \tag{6.11}$$

Chapter 7

Mobility models

Electron and hole mobility in semiconductor materials is affected by various external factors such as temperature, electric field, presence of defects, etc. These effects can be included while simulating devices in the DD simulator. In this chapter, available mobility models available in the the DD simulator are described together with the parameters.

All the mobility model parameters are set in `Mobility` part of the material config file.

7.1 Bulk mobility

Carrier mobility in the bulk of a low-doped semiconductor is affected by phonon scattering. Since phonon scattering increases with temperature, bulk mobility degrades with temperature. This is modeled by the following empirical relation.

$$\mu_b = \mu_{max} \left(\frac{T}{T_0} \right)^{-\alpha} \tag{7.1}$$

Here, μ_{max} is mobility of the low-doped semiconductor measured at temperature T_0, T is the operating temperature, and α is a constant determined from the measurements. All the above parameters can be set in `ConstantMob` section of material config file. The parameter values are different for electron and hole mobilties. Separate parameter

Table 7.1: Bulk mobility model parameters and their values

Symbol	Parameter	Electrons	Holes	Unit
μ_{max}	`mumax`	1450.0	450.0	cm^2/Vs
α	`alpha`	1.4	1.4	–

values are specified in the config file for electron and holes. The parameter names and their default values are listed in Table 7.1.

Note, that bulk mobility model is *always* active in DD simulations.

7.2 Doping dependence

Bulk mobility degradation is observed in doped semiconductors. This can be attributed to higher ionized impurity scattering in high doped regions. It has been modeled empirically by various research groups. An empirical model by Masetti et al is currently available in the DD simulator to model mobility degradation. It can be activated by specifying `Masetti` argument in the `DopingDep` part of `Mobility` section in global, material, or region-wise physics section as follows.

```
Physics: {
  Mobility: {
    DopingDep: { Masetti: []; }
  }
  ...
}
```

7.2.1 Masetti model

In the model proposed by Masetti, electron and hole mobility in the doped semiconductor is given by the following expression.

$$\mu_b = \mu_{min1} \cdot \exp\left(-\frac{P_c}{N_D}\right) + \frac{\mu_b - \mu_{min2}}{1 + (\frac{N_D}{C_r})^{\alpha}} - \frac{\mu_1}{1 + (\frac{C_s}{N_D})^{\beta}} \tag{7.2}$$

The parameters μ_{min1}, μ_{min2}, and μ_1 are mobility parameters, while μ_b is the bulk mobility in Eq 7.1. The mobility parameters, the doping

Table 7.2: Doping dependent mobility model parameters and their values

Symbol	Parameter	Electrons	Holes	Unit
μ_{min1}	`mumin1`	52.2	44.9	cm^2/Vs
μ_{min2}	`mumin2`	52.2	0.	cm^2/Vs
μ_1	`mu1`	43.4	29.0	cm^2/Vs
P_c	`Pc`	0	9.23×10^{16}	cm^{-3}
C_r	`Cr`	9.68×10^{16}	2.23×10^{17}	cm^{-3}
C_s	`Cs`	3.43×10^{20}	6.1×10^{20}	cm^{-3}
α	`alpha`	0.68	0.19	–
β	`beta`	2.0	2.0	–

parameters P_c, C_r, C_s and the exponents α, β are set in `Masetti` section in the material config file. The parameter names and their default values are listed in Table 7.2.

7.3 Mobility degradation in the channel

In a Metal-Oxide Semiconductor Field Effect Transistor (MOSFET), carrier mobility at insulator-semiconductor interface degrades due to various factors including field-induced quantum confinement, interface roughness, phonon scattering. Various semi-empirical or empirical models have been proposed in the literature to model this effect. In DD simulator, a model proposed by Lombardi et al [] is available to model mobility degradation at insulator-semiconductor interface. It can be activated by using the following syntax.

```
Physics: {
  Mobility: {
    FieldDep: { Lombardi: [];}
    ...
  }
  ...
}
```

7.3.1 Lombardi model

Contribution of surface acoustic phonons to the mobility ($\mu_{\rm ac}$) is accounted by the following expression.

$$\mu_{\rm ac} = \frac{B}{F_\perp} + \frac{C\left(\frac{N_D+N_2}{N_0}\right)^\lambda}{F_\perp^{1/3}\left(\frac{T}{300.0}\right)^k} \tag{7.3}$$

Here, $F_\perp$ is local electric field along the direction normal to the nearest insulator/semiconductor interface, B and C are the fitting parameters, N_0 and N_2 are reference doping parameters, λ and k are the exponents.

Similarly, contribution of surface roughness scattering to the mobility ($\mu_{\rm sr}$) is given by,

$$\mu_{\rm sr} = \left(\frac{(\frac{F_\perp}{F_{\rm ref}})^{\rm A*}}{\delta} + \frac{F_\perp^3}{\eta}\right)^{-1} \tag{7.4}$$

where, $F_{\rm ref} = 1$V/cm is the reference field parameter, δ and η are fitting parameters of the model, $F_\perp$ carries the same meaning as in Eq. 7.3.

The above two mobility contributions are combined with the bulk mobility μ_b calculated using Eq. 7.2 using Mattheisen's rule to obtain total low-field mobility.

$$\frac{1}{\mu_{\rm low}} = \frac{1}{\mu_b} + \exp\left(-\frac{d}{l_{\rm crit}}\right)\left(\frac{1}{\mu_{\rm ac}} + \frac{1}{\mu_{\rm sr}}\right) \tag{7.5}$$

Here, d is the distance from the nearest semiconductor/insulator interface and $l_{\rm crit}$ is a fitting parameter. The factor $\exp\left(-\frac{d}{l_{\rm crit}}\right)$ allows smooth transition from mobility degradation model at the interface to the bulk mobility.

The exponent $A*$ in Eq. 7.4 is given by the following expression.

$$A* = A + \frac{\alpha_\perp(n+p)}{(\frac{N_{\rm D}+N_1}{N_{\rm ref}})^\nu} \tag{7.6}$$

Parameter names in Eq. 7.3, Eq. 7.4, and Eq. 7.6 and their default values are listed in Table 7.3.

Table 7.3: Parameters for Lombard mobility degradation model and their default values

Symbol	Parameter	Electrons	Holes	Unit
B	`B`	4.75×10^7	9.925×10^6	cm/s
C	`C`	5.8×10^2	2.947×10^3	$\mathrm{cm}^{5/3}\mathrm{V}^{-2/3}\mathrm{s}^{-1}$
N_0	`N0`	1	1	cm^{-3}
N_1	`N1`	1	1	cm^{-3}
N_2	`N2`	1	1	cm^{-3}
λ	`lambda`	0.125	0.0317	–
k	`k`	1	1	–
δ	`delta`	5.82×10^{14}	2.0546×10^{14}	$\mathrm{cm}^2/\mathrm{Vs}$
A	`A`	2	2	$\mathrm{cm}^2/\mathrm{Vs}$
$\alpha_\perp$	`alpha`	0	0	cm^3
ν	`nu`	1	1	–
η	`eta`	5.82×10^{30}	2.0546×10^{30}	$\mathrm{V}^2\mathrm{cm}^{-1}\mathrm{s}^{-1}$
l_{crit}	`lcrit`	10^{-6}	10^{-6}	cm

7.4 High field saturation

Presence of high electric field in the carrier transport direction increases probability of phonon scattering which degrades carrier mobility. Thus, carrier drift velocity is not a linear function of electric field ($\vec{v}_{\text{drift}} \neq \mu\vec{F}$). This can be modeled by defining field dependent mobility, such that the linear relationship between $\vec{v}_d$ and $\vec{E}$ is reestablished ($\vec{v}_{\text{drift}} = \mu(\vec{F})\vec{F}$). Field dependent mobility model proposed by Canali et al is available in DD simulator to model this behavior at high electric fields. It can be activated as follows.

```
Physics: {
  Mobility: {
    HighFieldSat: { Canali = []; }
    ...
  }
  ...
}
```

7.4.1 Canali model

Canali model takes mobility μ_{low} at low electric field and updates it to model mobility degradation at high field. Mobility at high electric field parallel to the transport direction $F_{\|}$ is expressed as follows.

$$\mu(F_{\|}) = \frac{(\alpha+1)\mu_{\text{low}}}{\alpha + \left[1 + \left(\frac{(\alpha+1)\mu_{\text{low}}F_{\|}}{v_{\text{sat}}}\right)^{\beta}\right]^{1/\beta}} \tag{7.7}$$

Here,α is a fitting parameter that can be anywhere between 0 to 1. When $\alpha = 1$, Haensch [**?**] model is activated. The exponent β is temperature dependent and is given by the expression below.

$$\beta = \beta_0 \left(\frac{T}{300.0}\right)^{\beta_{\text{exp}}} \tag{7.8}$$

Similarly, saturation velocity v_{sat} in Eq. 7.7 is given below.

$$v_{\text{sat}} = v_{\text{sat0}} \left(\frac{T}{300.0}\right)^{v_{\text{satexp}}} \tag{7.9}$$

By default, electric field parallel to the transport direction $F_{\parallel}$ is calculated by projecting vector electric field $\vec{F}$ on the carrier transport direction.

$$F_{\parallel} = \vec{F} \cdot \frac{\vec{J}_{n/p}}{|\vec{J}_{n/p}|} \tag{7.10}$$

If `EParallel` is set to `GradQuasiFermi` in `HighFieldSat` section, then gradient of quasi Fermi level $\nabla\phi_{n/p}$ is used to calculate $F_{\parallel}$.

$$F_{\parallel} = \left|\nabla\phi_{n/p}\right| \tag{7.11}$$

Note, that μ_{low} in Eq. 7.7 combines all active mobility models (e.g. bulk mobility, doping dependence, normal field dependence, etc.) in the low field using Mattheisen's rule.

Chapter 8

Effect of mechanical stress

Semiconductor devices are subject to mechanical stress arising from the soldering, packaging, or fabrication processes. Mechanical stress results in deformation of the crystal-lattice, which changes lattice constant as well as semiconductor band-structure. The CB and VB edges are shifted. This causes wide range of changes in the material and the device data, such as threshold voltage, mobility, etc. Modeling the effects of mechanical stress on the semiconductor device characteristics is described below.

8.1 Stress-strain modeling

Applied stress on the semiconductor device deforms the crystal and introduces strain in it. This in-turn affects band-structure of the semiconductor. If the device-stress is provided as an input, strain is calculated using the following equation.

$$\vec{\epsilon} = C \cdot \vec{\sigma} \tag{8.1}$$

Here, strain-tensor and stress-tensor are written in 'engineering notation' as follows.

$$\vec{\epsilon}^{\mathrm{T}} = \begin{bmatrix} \epsilon_{xx} & \epsilon_{yy} & \epsilon_{zz} & \epsilon_{yz} & \epsilon_{xz} & \epsilon_{xy} \end{bmatrix} \tag{8.2}$$

$$\vec{\sigma}^{\mathrm{T}} = \begin{bmatrix} \sigma_{xx} & \sigma_{yy} & \sigma_{zz} & \sigma_{yz} & \sigma_{xz} & \sigma_{xy} \end{bmatrix} \tag{8.3}$$

A strain tensor, when represented in full matrix form, is given below.

$$\bar{\bar{\epsilon}} = \begin{bmatrix} \epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{xy} & \epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{xz} & \epsilon_{yz} & \epsilon_{zz} \end{bmatrix} \tag{8.4}$$

However, for stress-strain calculations, engineering format is used.

The compliance matrix (C) is a 6×6 matrix which defines relationship between stress and strain. Compliance matrix for a cubic crystal symmetry is given below.

$$C = \begin{bmatrix} C_{11} & C_{12} & C_{12} & 0 & 0 & 0 \\ C_{12} & C_{11} & C_{12} & 0 & 0 & 0 \\ C_{12} & C_{12} & C_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 & C_{44} \end{bmatrix} \tag{8.5}$$

Values of the material parameters S_{11}, S_{12}, S_{44} are provided in `ComplianceMat` section of `MaterialProp` part of material config file with the keywords `S11`, `S12`, and `S44`, respectively. Their unit is "cm^2/dyne" (1 m^2/Pa = 10 cm^2/dyne).

Note, that conventionally compressive strain is *negative* and tensile strain is *positive*. This convention is also followed in DD solver.

Values of device-stress *or* device-strain are specified by the user in engineering format as a list of 6 numbers in the `Physics` section. If stress is provided, strain is calculated from stress using Eq. 8.1. Alternately, user can provide device-strain directly. Stress/strain is defined in global, material-wise, or region-wise `Physics` sections in `StressEffects` part in the DD solver config file as follows.

```
StressEffects:
{
    DeformationPotModel = ["BandMin"]
    Stress = [0,0,0,0,0,0]
    Strain = [0,0,0,0,0,0]
    CrystalX = [1,0,0]
    CrystalY = [0,1,0]
}
```

As shown above, the user-provided constant stress(strain) is specified with `Stress`(`Strain`) as a list of 6 components in engineering notation (see Eq. 8.3 or 8.2). If the device coordinate system (in which stress/strain is defined) is different from the crystal coordinates, then X- and Y- axes of the crystal in device coordinate system can be specified with the keywords `CrystalX` and `CrystalY`.

8.2 Band-edge shift

8.2.1 CB-edge shift

Silicon has six CB valleys - three on the positive side of each major axis ($\Delta_x, \Delta_y, \Delta_z$) and the other three on the negative side in the reciprocal-lattice space. All these CB valleys are degenerate (i.e. they have the same energy minimum). Applying compressive stress increases energy of the valley-minima, while tensile stress decreases it. Applying anisotropic stress shifts different valleys by different energy, due to symmetry reason. This energy shift per valley in the case of Silicon is given by.

$$\Delta E_{\mathrm{C,i}} = \Xi_d \cdot (\epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}) + \Xi_u \cdot \epsilon_{ii} \cdots \text{where}\, i = x, y, z \quad (8.6)$$

Here, Ξ_d, Ξ_u are CB deformation potentials and $\Delta E_{\mathrm{C,i}}$ is energy shift of the valley at $i = x, y, z$. Due to symmetry reasons, shear components of strain (i.e. $\epsilon_{yz}, \epsilon_{xz}, \epsilon_{xy}$) do not affect CB energy shift (at least in first order approximation).

The parameters Ξ_d and Ξ_u are defined in `StressEffect` section of the `BandStructure` part of the material config file. They are specified with the keyword `Xu` and `Xd`, respectively.

After calculating the CB edge shifts for all the valleys, CB edge is obtained as follows. If the keyword `BandMin` is specified with the argument `DeformationPotModel`, then CB edge is set by adding minimum of all the CB edge shifts to it.

$$E_{\mathrm{C,strained}} = E_{\mathrm{C,relaxed}} + \min_i \Delta E_{\mathrm{C,i}} \tag{8.7}$$

If no keyword is specified, then average CB edge shift is obtained by the following equation.

$$\Delta E_{\mathrm{C}} = -\log\left(\frac{1}{N}\sum_i^N \exp\left(-\frac{\Delta E_{\mathrm{C,i}}}{kT}\right)\right) \tag{8.8}$$

The above equation performs a 'weighted average' of CB valley shifts with electron concentration in each valley as 'weighing factor'.

8.2.2 VB-edge shift

VB maximum of Silicon is doubly degenerate, with two bands - Light Hole (LH) and Heavy Hole (HH) band - having the energy maxima at the Γ-point in the reciprocal-lattice space. Similar to the CB, applying compressive stress increases energy of the valley-maxima, while tensile stress decreases it. Due to the symmetry properties of the LH and HH bands, application of strain energetically separates LH and HH bands and breaks degeneracy of the VB maxima. LH and HH energy shifts are derived from k.p theory without considering spin-orbit split-off band. They are given by.

$$\Delta E_{\mathrm{V,hh}} = a \cdot (\epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}) + \delta E \tag{8.9}$$

$$\Delta E_{\mathrm{V,lh}} = a \cdot (\epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}) - \delta E \tag{8.10}$$

$$\delta E = \sqrt{\frac{b^2}{2}((\epsilon_{xx} - \epsilon_{yy})^2 + (\epsilon_{yy} - \epsilon_{zz})^2 + (\epsilon_{zz} - \epsilon_{xx})^2) + d^2(\epsilon_{xy}^2 + \epsilon_{yz}^2 + \epsilon_{xz}^2)} \tag{8.11}$$

Here, a, b, and d are k.p parameters. If Δ_{so} $(E_{\mathrm{V,hh/lh}} - E_{\mathrm{so}})$ is small enough to not neglect it, then the 3 band k.p equation is solved at Γ-point to obtain energy shifts of all the bands.

The parameters a, b and d are defined in `StressEffect` section of the `BandStructure` part of the material config file. They are specified with the keywords `akp`, `bkp`, and `dkp`, respectively.

After calculating the VB edge shifts for all the bands, VB edge is obtained as follows. If the keyword `BandMin` is specified with the argument `DeformationPotModel`, then VB edge is set by adding maximum of all the VB edge shifts to it.

$$E_{\mathrm{V,strained}} = E_{\mathrm{V,relaxed}} + \min_{\mathrm{i=lh,hh,so}} \Delta E_{\mathrm{V,i}} \tag{8.12}$$

If no keyword is specified, then average VB edge shift is obtained by the following equation.

$$\Delta E_{\mathrm{V}} = \log\left(\frac{1}{3} \sum_{\mathrm{i=lh,hh,so}} \exp\left(\frac{\Delta E_{\mathrm{V,i}}}{kT}\right)\right) \tag{8.13}$$

Similar to the CB edge, the above equation performs a 'weighted average' of VB valley shifts with *hole* concentration in each valley as 'weighing factor'.

8.3 Mobility modification

Net e/h mobility is a weighted average of e/h mobility in each of the CB/VB valleys, weighted by the e/h occupancy of that valley.

Application of strain breaks degeneracy of both CB and VB of Silicon. As a result, electron/hole re-population takes place. For example, electrons populate all the six CB valleys equally in relaxed Si. In strained Silicon, more electrons occupy the CB valley(s) with lower energy. As a result, net μ_e changes on application of strain. This is accounted in the DD simulations by modifying the mobility scaling factor (μ_r).

Each of the CB valleys has an ellipsoid shape, which results in effective mass anisotropy. This in-turn results in anisotropic μ_r. Anisotropic μ_r for each of the Silicon CB valleys is given below.

$$\overline{\overline{\mu}}_{\mathrm{r,x}} = \begin{bmatrix} m_c/m_l & 0 & 0 \\ 0 & m_c/m_t & 0 \\ 0 & 0 & m_c/m_t \end{bmatrix} := \mathrm{diag}(m_c/m_l, m_c/m_t, m_c/m_t) \tag{8.14}$$

$$\overline{\overline{\mu}}_{\mathrm{r,y}} = \mathrm{diag}(m_c/m_t, m_c/m_l, m_c/m_t) \tag{8.15}$$

$$\overline{\overline{\mu}}_{\mathrm{r,z}} = \mathrm{diag}(m_c/m_t, m_c/m_t, m_c/m_l) \tag{8.16}$$

Here, $\frac{1}{m_c} = \frac{2}{m_t} + \frac{1}{m_l}$ is the average electron effective mass in relaxed Silicon.

An average $\overline{\overline{\mu}}_{\mathrm{r,avg}}$ is obtained from $\overline{\overline{\mu}}_{\mathrm{r}}$ of all the valleys using the following equation.

$$\overline{\overline{\mu}}_{\mathrm{r,avg}} = \frac{\sum_i \overline{\overline{\mu}}_{\mathrm{r,i}} \cdot \exp\left(-\Delta E_{\mathrm{C,i}}/kT\right)}{\sum_i \exp\left(-\Delta E_{\mathrm{C,i}}/kT\right)} \tag{8.17}$$

Scalar mobility enhancement factor (μ_r), which scales mobility, is obtained from Ref. 8.14 and electron current direction ($\hat{J}_n = \frac{\vec{J}_n}{|\vec{J}_n|}$) at each vertex in the device as follows.

$$\mu_r(\vec{r}) = \hat{J}_n(\vec{r})^{\mathrm{T}} \cdot \overline{\overline{\mu}}_{\mathrm{r,avg}} \cdot \hat{J}_n(\vec{r}) \tag{8.18}$$

At each vertex in the device, mobility is scaled by the above factor $\mu_r(\vec{r})$. In this way, electron mobility modification due to electron re-population is accounted in DD simulations.

In the similar fashion, hole mobility scaling factor is calculated by performing weighted averaging over LH, HH, and spin-orbit bands. Note, that effective masses and thus mobility of the hole bands is assumed to be isotropic.

Electron longitudinal and transverse effective mass values m_l and m_t are defined in `StressEffect` section of the `BandStructure` part of the material config file. They are specified with the keywords `mc_l`, `mc_t`. Similarly, effective masses of LH, HH, and spin-orbit bands are specified in the same section with the keywords `mv_lh`, `mv_hh`, and `mv_so`, respectively.

Note: Strain-induced modification of the effective mass is not accounted for in the above model.

Chapter 9

Generation-Recombination models

Electron and hole continuity equations (Eq. 5.4) includes net recombination rates of the carriers. Total generation rate of all the generation processes is subtracted from the total recombination rate of of all the recombination processes to calculate net recombination rate of the carriers. This chapter explains all the generation and recombination processes that can be activated in the DD simulator.

9.1 Shockly-Read-Hall recombination

Schockley-Read-Hall (SRH) recombination takes place by capture of an electron from the CB by a defect level and its subsequent emission to the VB resulting in annihilation of a hole. Similarly, generation takes place when an electron can be captured by the defect level *from the VB* creating a hole in the VB. It is subsequently emitted to the CB. Both the above processes can be activated in DD simulations as follows.

```
Physics: {
```

```
  Recombination: {
    SRHRecombination = ["DopingDep"];
  }
}
```

Recombination rate due to SRH process is calculated by using the following equations.

$$R_{\text{srh}} = \frac{n \cdot p - n_i^2}{(n + n1) \cdot \tau_p + (p + p1) \cdot \tau_n} \tag{9.1a}$$

$$n1 = n_i \cdot \exp\left(\frac{qE_{\text{trap}}}{kT}\right) \tag{9.1b}$$

$$p1 = n_i \cdot \exp\left(-\frac{qE_{\text{trap}}}{kT}\right) \tag{9.1c}$$

$$\tag{9.1d}$$

In the above equations, n and p are electron and hole densities, n_i is intrinsic carrier density, E_{trap} is the dominant trap energy level in the semiconductor (measured from the mid-gap, positive value shifts trap levels towards the CB), $\frac{kT}{q}$ is the thermal voltage, τ_p and τ_n are electron and hole life-times (in sec).

9.1.1 Carrier lifetimes

Electron and hole lifetimes ($\tau_{n/p}$) depend on various external factors.

Doping dependence

In DD simulator, e/h lifetime dependence on doping concentration is modeled by the following expression.

$$\tau_{n/p} = \tau_{\text{min,n/p}} + \frac{\tau_{\text{max,n/p}} - \tau_{\text{min,n/p}}}{1 + \left(\frac{N_D}{N_{\text{ref,n/p}}}\right)^{\gamma_{n/p}}} \tag{9.2}$$

Here, $\tau_{\text{min,n/p}}$, $\tau_{\text{max,n/p}}$ are fitted parameters of carrier lifetimes, N_D is net doping concentration, and $N_{\text{ref,n/p}}$ is reference doping parameter. Doping dependence can be activated by specifying `DopingDep` keyword in `SRHRecombination` keyword list.

Table 9.1: SRH Scharfetter model parameters and their values

Symbol	Parameter	Electrons	Holes	Unit
$\tau_{\max}$	`taumax`	10^{-6}	10^{-6}	sec
$\tau_{\min}$	`taumin`	10^{-9}	10^{-9}	sec
N_{ref}	`Nref`	10^{16}	10^{16}	cm^3
γ	`Gamma`	1.0	1.0	–
E_{trap}	`Etrap`	0.0		eV

Parameters are defined in `SRHRecombination` section of the `Recombination` part of the material config file. Separate parameters for electrons and holes are defined by using lists as described in Section **??**. Mole fraction dependence of the parameters can be modeled as described in Section 2.8.1. Default values of the above parameters are provided in Table 9.1

9.2 Band-to-band generation

In the presence of high electric field, electrons undergo tunneling from the VB states to the CB states. This generates an electron-hole pair in the semiconductor. This is called Band-to-band Tunneling (BTBT) process. It can be activated in DD simulations as follows.

```
Physics: {
  Recombination: {
    Band2BandGen = [ ];
  }
}
```

The expression for Band-to-band generation rate can be evaluated analytically for a homogeneous semiconductor subject to a constant electric field. Different theoretical analyses lead to slightly different powers of the electric field. Generation rate due to BTBT process can

Table 9.2: Band-to-band generation model parameters and their values

Symbol	Parameter	Values	Unit
A_1	`A1`	10^{-6}	sec
A_{1p5}	`A1p5`	10^{-9}	sec
A_2	`A2`	10^{16}	cm^3
B	`B`	1.0	–
Model type	`ModelType`	2	–

be modeled in the DD simulator by selecting any one of the following expressions.

$$G_{\text{btb}} = \begin{cases} A_1 \cdot |F| \cdot \exp\left(-\frac{B}{|F|}\right) & \dots \text{ model} = 2 \\ A_{1p5} \cdot |F|^{1.5} \cdot \exp\left(-\frac{B}{|F|}\right) & \dots \text{ model} = 3 \\ A_2 \cdot |F|^2 \cdot \exp\left(-\frac{B}{|F|}\right) & \dots \text{ model} = 4 \end{cases} \tag{9.3}$$

Here, A_1, A_{1p5}, A_2, B are fitting parameters of the models. $|F|$ is the magnitude of electric field.

Parameters in Eq. 9.3 are defined in `BandToBandGen` section of the `Recombination` part of the material config file. Their default values are provided in Table 9.2. The parameter `ModelType` specifies which of the models in Eq. 9.3 is used for BTBT rate.

9.3 Fowler-Nordheim tunneling

High electric-field perpendicular to the channel creates a triangular barrier in the oxide normal to the channel direction. The "tunnel length" is small at the top of the triangular barrier leading to tunneling few high energy electrons from CB of Silicon to the oxide. If the very high oxide electric field is arising from high positive gate-voltage, tunnelled electrons are transported to the nearby poly-gate. If the high oxide field is arising from high negative gate-voltage, electrons are tunnelled from the gate-poly into Silicon channel. Both these physical phenomena generate gate current flow away from the gate or to the gate, respectively. This process is called "Fowler-Nordheim" tunneling.

If the poly-gate is connected to the external contact, this gate current flows through the external circuit. If the poly-gate is a floating gate, these electrons are "trapped" into the gate causing charge pile-up in the gate lowering its electrostatic potential. This mechanism is used in flash memory devices to store the logic '1' (electrons are trapped in the floating poly-gate) or '0' (electrons are not trapped). Applying a high positive gate voltage is called the 'write' step, whereas applying a high negative gate voltage to flush out electrons from the floating gate is called the 'read' step.

This tunneling is modeled in the DD Solver by a general equation given by Eq. 9.4.

$$G_{\rm fn}^{\rm surf}(\vec{r}) = sign(\vec{F}_{\perp}\vec{r}) \cdot \gamma \cdot A \cdot \exp\left(-\frac{B}{|\vec{F}(\vec{r}) \cdot \hat{n}_{\rm surf}|}\right) \tag{9.4}$$

Here, $G_{\rm fn}^{\rm surf}$ is electron generation rate at the Semiconductor/oxide interface. $\vec{F}_{\perp}(\vec{r})$ is electric field along the interface normal ($\vec{F}(\vec{r}) \cdot \hat{n}_{\rm surf}$) pointing towards oxide region. $sign(x)$ is a sign function. The parameters A and B are defined as follows.

$$A(\vec{r}) = \begin{cases} A_{\rm wr} & \ldots \vec{F}_{\perp}(\vec{r}) < 0 \\ A_{\rm er} & \ldots \text{otherwise} \end{cases} \tag{9.5}$$

$$B(\vec{r}) = \begin{cases} B_{\rm wr} & \ldots \vec{F}_{\perp}(\vec{r}) < 0 \\ B_{\rm er} & \ldots \text{otherwise} \end{cases} \tag{9.6}$$

These parameters are defined in `Fowler` section of the `Recombination` part of the material config file. Their default values are provided in Table 9.3.

Generation rate given by Eq. 9.4 is added to the total electron surface generation rate at $\vec{r}$.

Fowler-Nordheim tunneling is activated using the following syntax.

```
Physics*Material*Silicon/Oxide: {
  Recombination: {
    Fowler = [ HoleTun ];
  }
}
```

Table 9.3: Fowler-Nordheim model parameters and their values

Symbol	Parameter	Values	Unit
A_{wr}	`Awrite`	10^{-6}	$1/\text{V}^2$
A_{er}	`Aerase`	10^{-9}	$1/\text{V}^2$
B_{wr}	`Bwrite`	10^{-6}	V/cm
B_{er}	`Berase`	10^{-9}	V/cm
γ	`Gmult`	1	1

If the `HoleTun` is added to the keywords, the hole tunneling is activated instead of electron tunneling. Thus, generation rate given by Eq. 9.4 is added to the total hole surface generation rate at $\vec{r}$.

9.4 Impact ionization

In the presence of extremely high electric field, electrons are sufficiently accelerated to eject more electrons from the VB to the CB, thereby creating an electron-hole pair in the semiconductor. This is called impact ionization process. The generated electrons also undergo acceleration and subsequent collision with electrons in VB generating more e-h pairs. Thus, a single electron entering in a high field region can create a large number of electrons. Similar process takes place for holes.

Generation rate for the impact ionization process can be expressed as follows.

$$G_{II} = \alpha_n \left|\vec{J}_n\right| + \alpha_p \left|\vec{J}_p\right| \tag{9.7}$$

Here, $\left|\vec{J}_n\right|$ and $\left|\vec{J}_p\right|$ are magnitudes of e and h current densities while α_n and α_p are ionization coefficients of e and h, respectively.

Impact ionization can be activated in DD simulations as follows.

```
Physics: {
  Recombination: {
    AvalancheGen = ["vanOverstraeten" ];
  }
}
```

The keyword `vanOverstraeten` specifies that 'Van Overstraeten' model is used for the calculation of the ionization coefficients.

9.4.1 Van Overstraeten model

The ionization coefficients α_n and α_p strongly depend on the electric field parallel to the carrier transport direction. It is modeled using the following expression.

$$\alpha_{n/p}(F_{\mathrm{ava,n/p}}) = \gamma \cdot a_{n/p} \cdot \exp\left(-\frac{\gamma b_{n/p}}{F_{\mathrm{ava,n/p}}}\right) \tag{9.8}$$

Here, $a_{n/p}$ and $b_{n/p}$ in Eq. 9.8 are the model parameters, and F_{ava} is magnitude of the electric field parallel to the carrier transport direction.

$$F_{\mathrm{ava,n/p}} = \vec{F} \cdot \frac{\vec{J}_{n/p}}{|\vec{J}_{n/p}|} \tag{9.9}$$

The factor γ is given as follows.

$$\gamma = \frac{\tanh\left(\frac{\hbar\omega_{\mathrm{op}}}{2kT_0}\right)}{\tanh\left(\frac{\hbar\omega_{\mathrm{op}}}{2kT}\right)} \tag{9.10}$$

Here, $\hbar\omega_{\mathrm{op}}$ is the optical phonon energy.

Parameters in Eq. 9.8 and Eq. 9.10 are defined in `VanOverstraeten` section of the `Recombination` part of the material config file. Default values of the parameters are provided in Table 9.4.

Notice, that there are two values corresponding to the parameters a and b which are noted as `ah`, `al` and `bh`, `bl`, respectively. They correspond to low field and high field values of `a` and `b`, respectively.

9.5 Optical generation

Electron-hole pairs can be generated by irradiating the semiconductor with photons of higher energy than the band gap. The DD simulator provides various analytical models to mimic this effect.

Table 9.4: Van Overstraeten model parameters and their values

Symbol	Parameter	Electron	Hole	Unit
a	`al`	7.03×10^5	1.582×10^6	–
	`ah`	7.03×10^5	6.71×10^5	–
b	`bl`	1.231×10^6	2.036×10^6	–
	`bh`	1.231×10^6	2.036×10^6	–
F_0	`F0`	4×10^5	4×10^5	V/cm
$\hbar\omega_{\text{op}}$	`hbarOmega`	0.063	0.063	eV
T_0	`T0`	300.0		K

9.5.1 Constant generation

Constant generation model sets generation rate of e-h pairs to a constant value. This model can be activated in a specific region or the material as follows.

```
Physics: {
  Recombination: {
    ConstantGeneration = ["Rate = 1E10", "TimeRamp"];
  }
}
```

The keyword `Rate = 1E10` specifies constant rate of 10^10cm^{-3}, while `TimeRamp` specifies that the rate will be changed from initial value of 0 cm^{-3} to the final value specified in the `Quasistationary` or `Transient` group in the following format.

```
Ramp: {
  Voltage*gate = 1.0;
  ConstantGeneration*Rate = 1E15;
}
```

9.5.2 Generation from file

Optical generation rate can also be input from maximum 20 files generated from the Finite Difference Time Domain (FDTD), Beam Propagation Method (BPM), or mode simulations. The name of the files is specified by arguments `OpticalGenerationFile1`... `OpticalGenerationFile20` in the device `File` section. The name of the physical quantity to be read from the files is specified by the argument `OpticalGenerationField`.

```
File: {
  OpticalGenerationFile1 = "fdtdFinFET_TimeAvg_AvgMid_fdtd.h5";
  OpticalGenerationField = "AbsElectricField";
}
```

Optical generation from the files can be activated by adding `GenerationFromFile` keyword in `Recombination` section as follows.

```
Physics: {
  Recombination: {
    GenerationFromFile = ["FileIds = 1", "Scaling = 0", "TimeRamp"];
  }
}
```

The keyword `FileIds = 1` specifies the file id from which the generation rate is read. In this case the file is specified by argument `OpticalGenerationFile1`. Keyword `Scaling = 0` specifies the scaling factor for the generation rate read from the file. `TimeRamp` specifies that the scaling factor will be changed from initial value of 0 to the final value specified in the `Quasistationary` or `Transient` group in the following format.

```
Ramp: {
  Voltage*gate = 1.0;
  GenerationFromFile*Scaling = 1;
}
```

9.5.3 Approximate Radiative Recombination

Radiative recombination in the LEDs can be modeled in the DD simulator approximately by using the approximate radiative recombination model. It can be activated in the simulations as follows.

```
Physics: {
  Recombination: {
    ApproxRadiativeRec = [ ];
  }
}
```

The model calculates the recombination rate by the following expression.

$$R = C \cdot n \cdot p \cdot \left(\frac{T}{T_{\text{par}}} \right)^{\alpha} \tag{9.11}$$

Here, C is the model rate parameter, α is the temperature exponent, and T_{par} is the reference temperature. Note, that due to the highly approximate nature of this model, for predictive analysis, it is recommended to calibrate the parameters for each individual device structure.

Parameters in Eq. 9.11 are defined in `RadiativeRec` section of the `Recombination` part of the material config file.

Chapter 10

Traps

Various crystal defects may be present in the material or at material interfaces. These defects act as recombination centers. The defects can trap electrons during device operation and get charged. Charged defects can affect electrostatics of the device. These effects can be modeled in the DD simulator by defining the trap concentration in the material/region or at the material/region interfaces.

10.1 Trap types

10.1.1 Donor type traps

Donor type traps are neutral when unoccupied, and carry a unit positive electron charge when occupied. They are set by specifying keyword `Donor` in the `Type`.

10.1.2 Acceptor type traps

Acceptor type traps are neutral when unoccupied, and carry a unit negative electron charge when occupied. They are set by specifying keyword `Acceptor` in the `Type`.

10.2 Defining Bulk Traps

Bulk traps can be defined in the material/region physics section of the config file in the `Traps` subsection as follows.

```
Physics*Material*Silicon: {
  Traps: {
    MidGapTr1: {
      Type = ["Donor", "Gaussian", "FromValenceBand"];
      Conc = 1E17; Energy = 0.0; EnSigma = 0.2;
    }
  }
}
```

Each subsection in `Traps` subsection creates a new trap definition. Each trap definition consists of a set of arguments. Additionally, the argument `Type` lists a list of comma separated keywords which sets a number of internal flags. The `MidGapTr1` definition in the above example creates `Donor` traps with energetic distribution of `Gaussian` in all the regions of Silicon material. `FromValenceBand` specifies that the peak energy value is set from the VB edge (positive value means energy into the band gap). In the `Trap` section, the argument `Conc` specifies peak concentration of defects, `Energy` sets the energy of the peak from the VB edge, and `EnSigma` sets the standard deviation of the energetic Gaussian.

10.3 Energetic trap distributions

Energetic trap distribution in the band gap of the semiconductor is often approximated by analytic shapes. Following shapes of energetic trap distributions are available in DD simulator.

- `Gaussian`: A Gaussian distribution is defined by setting the keyword `Gaussian` in `Type`. It is given by,

$$D_{\text{it}}(E) = D_{\text{it0}} \cdot \exp\left(-\frac{1}{2}\left(\frac{E - E_0}{\sigma}\right)^2\right)$$

- `Uniform`: A uniform distribution is defined by setting the keyword `Uniform` in `Type`. It is given by,

$$D_{\rm it}(E) = \begin{cases} D_{\rm it0} & \text{if } E_0 - \frac{1}{2}\sigma < E < E_0 + \frac{1}{2}\sigma \\ 0 & \text{otherwise} \end{cases}$$

- `Exponential`: An exponential distribution is defined by setting the keyword `Exponential` in `Type`. It is given by,

$$D_{\rm it}(E) = D_{\rm it0} \cdot \exp\left(-\frac{|E - E_0|}{\sigma}\right)$$

- `Level`: A single level distribution is defined by setting the keyword `Level` in `Type`. It is given by,

$$D_{\rm it}(E) = D_{\rm it0} \cdot \delta(E - E_0)$$

- `Table`: Energetic trap distribution can be specified as a piecewise linear curve. It can be set by specifying argument `ConcTable` with a list of floating point numbers. In this list, each trap energy level is followed by the trap concentration at that level. For example a piecewise linear trap distribution with $D_{\rm it}(-0.25) = 1E13$, $D_{\rm it}(0.) = 1E15$, and $D_{\rm it}(0.25) = 1E13$ is created by specifying the following argument in the trap definition.

```
ConcTable = [-0.25, 1E13, 0.0, 1E15, 0.25, 1E13];
```

In all the above trap distributions, E_0 in eV is specified by `Energy` and σ in eV is set by `EnSigma`. $D_{\rm it0}$ is set by `Conc`. For `Level` traps, `Conc` has the unit of $\rm cm^{-3}$ for bulk traps, and in $\rm cm^{-2}$ for interface traps. For all other distributions, `Conc` has the unit of $\rm cm^{-3}eV^{-1}$ for bulk traps, and in $\rm cm^{-2}eV^{-1}$ for interface traps.

Except `Level` trap, all energetic trap distributions create 13 trap energy levels in the band gap. Concentration at each of the energy levels creates the energetic distribution with the given shape. `Level` trap creates exactly one trap level at the energy `E0`.

By default, the trap energy levels given by `E0` or in `Table` distribution are defined relative to the mid-band energy. Specifying the

keyword `FromValenceBand` in `Type` defines trap energy levels from the VB in the direction of the CB edge. Specifying the keyword `FromConductionBand` defines trap energy levels from the CB in the direction of the VB edge.

Specifying `ClipTrapsToBandgap` in `Type` clips energetic trap distribution within the band gap.

10.4 Spatial trap distribution

In addition to the energetic distribution, spatially distributed trap densities can also be created in DD simulator. Following types of spatial distributions are available.

- `SpatiallyGaussian`: A spatially Gaussian distribution is defined by setting the keyword `SpatiallyGaussian` in `Type`. It is given by,

$$D_{\mathrm{it}}(E,x,y,z) = D_{\mathrm{it}}(E)\cdot\exp\left(-\frac{1}{2}\left(\frac{x-x_0}{\sigma_x^s}\right)^2+\left(\frac{y-y_0}{\sigma_y^s}\right)^2+\left(\frac{z-z_0}{\sigma_z^s}\right)^2\right)$$

- `SpatiallyExponential`: A spatially exponential distribution is defined by setting the keyword `SpatiallyExponential` in `Type`. It is given by,

$$D_{\mathrm{it}}(E,x,y,z) = D_{\mathrm{it}}(E)\cdot\exp\left(-\frac{|x-x_0|}{\sigma_x^s}-\frac{|y-y_0|}{\sigma_y^s}-\frac{|z-z_0|}{\sigma_z^s}\right)$$

- `SpatiallyUniform`: A spatially uniform distribution is defined by setting the keyword `SpatiallyUniform` in `Type`. It is given by,

$$D_{\mathrm{it}}(E,x,y,z) = \begin{cases} D_{\mathrm{it}}(E) & \text{if} |x-x_0|<\sigma_x^s \cap |y-y_0|<\sigma_y^s \cap |z-z_0|<\sigma_z^s \\ 0 & \text{otherwise} \end{cases}$$

- `SpatiallyLocalized`: A spatially localized distribution is defined by setting the keyword `SpatiallyLocalized` in `Type`. It is given by,

$$D_{\rm it}(E,\vec{r}) = \begin{cases} D_{\rm it}(E) & \text{if } \vec{r} = \left\{ \vec{r}_v : v = \underset{i \in \text{vertices}}{\text{argmin}} \, |\vec{r}_i - \vec{r}_0| \right\} \\ 0 & \text{otherwise} \end{cases}$$

  The above equation essentially sets a single trap level at the vertex nearest to $\vec{r}_0$.

In all the above trap distributions, $\vec{r}_0$ is specified in μm by `Center` and $\vec{\sigma}_s$ in μm is set by `Sigma`. $D_{\rm it}(E)$ is calculated at each energy level E by the energetic distribution.

10.5 Trapping and de-trapping models

During device operation, various trapping/de-trapping processes take place simultaneously. Rates of these processes determine occupancy of the traps.

- Electrons in the CB are captured by the traps with the rate of $c^n_{\rm C}$. This process charges an `Acceptor` trap, or it neutralizes positive charge on a charged `Donor` trap.

- Holes in the VB are captured by the traps with the rate of $c^p_{\rm V}$. This results in charging of a `Donor` trap, or a neutralization of a charged `Acceptor` trap.

- The trapped electrons are emitted to the CB with the rate of $e^n_{\rm C}$. This process neutralizes an `Acceptor` trap, or it creates a positive charge on a neutral `Donor` trap.

- The trapped holes are emitted to the VB with the rate of $e^p_{\rm V}$. This process neutralizes a `Donor` trap, or it charges a neutral `Acceptor` trap.

10.5.1 Trap occupancy

Occupancy of the trap levels depends on the rates of charging and discharging of the traps. Electron occupation factor f^n of the trap level is determined by net rate of capture of CB electrons r_C^n minus net rate of capture of VB holes r_V^p. This is given by,

$$\frac{\partial f^n}{\partial t} = r_C^n - r_V^p \tag{10.1a}$$

$$r_C^n = (1 - f^n) \cdot c_C^n - f^n \cdot e_C^n \tag{10.1b}$$

$$r_V^p = (1 - f^p) \cdot c_V^p - f^p \cdot e_V^p \tag{10.1c}$$

$$f^p = 1 - f^n \tag{10.1d}$$

If the capture and emission rates are known, then the set of Eq. 10.1 can be written as a partial differential equation in f^n. In `Quasistationary` ramps, the steady state is reached at each step and $\frac{\partial f^n}{\partial t} = 0$. This condition, when inserted in Eq. 10.1, yields electron occupancy at a trap level f^n as follows.

$$f^n = \frac{c_C^n + e_V^p}{c_C^n + c_V^p + e_C^n + e_V^p} \tag{10.2}$$

Once f^n is known, f^p is calculated using Eq. 10.1d. Trapped charge density on the `Acceptor` traps is given by $-f^n \cdot D_{\text{it,acc}}$, where negative sign implies a negatively charged trap. Trapped charge density on the `Donor` traps is given by $f^p \cdot D_{\text{it,donor}}$. Net trapped charged density is taken into account in solving *Poisson* equation.

Note, that the set of Eq. 10.1 are solved for each individual trap energy level to calculate occupancy of that level. For example, if there are 13 trap levels in the band gap, occupancy of each of the levels f^n will be calculated by solving Eq. 10.1 for each level separately.

10.5.2 Capture and emission rates

Capture and emission rates used in the above equation depend on various factors including trap energy level, carrier density, and the

current density. Capture rate of an electron from the CB and that of a hole from the VB is given by,

$$c_C^n = \sigma_n \left((1 - g_n^J) \cdot v_{\text{th}}^n \cdot n + g_n^J \cdot \frac{|\vec{J}_n|}{q} \right) \tag{10.3a}$$

$$c_V^p = \sigma_p \left((1 - g_p^J) \cdot v_{\text{th}}^p \cdot p + g_p^J \cdot \frac{|\vec{J}_p|}{q} \right) \tag{10.3b}$$

Here, $\sigma_{n/p}$ are electron/hole capture cross-sections, v_{th}^n and v_{th}^p are thermal velocities of electron and hole. The model parameters $g_{n/p}^J$ determine proportionate contribution of n and of $|\vec{J}_n|$ to the total capture rate.

$$e_C^n = \sigma_n \cdot v_{\text{th}}^n \cdot n1/g_n + e_{C0}^n \tag{10.4a}$$

$$e_V^p = \sigma_p \cdot v_{\text{th}}^p \cdot p1/g_p + e_{V0}^p \tag{10.4b}$$

$$n1 = N_C \exp\left(\frac{q(E_{\text{trap}} - E_C)}{kT} \right) \tag{10.4c}$$

$$p1 = N_V \exp\left(\frac{q(E_V - E_{\text{trap}})}{kT} \right) \tag{10.4d}$$

where, $g_{n/p}$ are degeneracy factors, E_{trap} is the trap energy level, and $N_{C/V}$ are CB and VB DOS. An additive constants to the emission rates are given by e_{C0}^n for electron emission and e_{V0}^p for hole emission.

When $g_{n/p}^J = 0$ and $e_{\text{C0}}^n = e_{\text{V0}}^p = 0$, the above equations follow the principle of detailed balance.

The above defined parameters can be set in `Traps` section of `Recombination` part of material config file. Their default values are provided in Table 10.1.

10.6 Config file of diode with traps

An example DD solver config file for simulating a diode with traps is given below. As shown in Fig. 10.1, the diode consists of two Silicon regions named – `RegSi1` and `RegSi2`. Interface traps have been defined at the interface between these two regions.

Table 10.1: Parameters of trap capture and emission rates and their default values

Symbol	Parameter	Electron	Hole	Unit
v_{th}	`vth0`	2.3×10^6	1.7×10^6	cm/sec
g^J	`Jfactor`	0	0	–
g	`Gfactor`	2	2	–
σ	`Xsection`	10^{-15}	10^{-15}	cm^2
$e_{\text{C0/V0}}$	`ConstEmissionRate`	0	0	$\text{cm}^{-3}\text{sec}^{-1}$

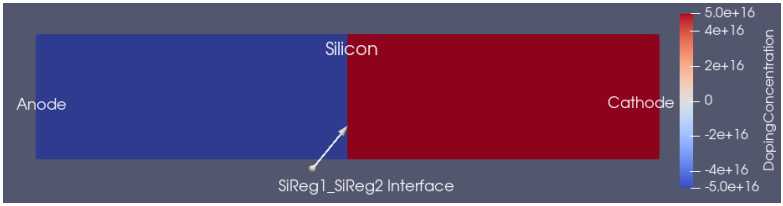


Figure 10.1: A 2D diode structure with two Silicon regions 'RegSi1' and 'RegSi2' is simulated by defining traps at 'SiReg1_SiReg2' interface.

```
File: {
  Device = "diode_str.cfg";
  Out = "Diode";
  Simulation = "DD";
}

Contacts: {
  Anode: {Voltage = 0.0; Type = ["Semiconductor"];}
  Cathode: {Voltage = 0.0; Type = ["Semiconductor"];}
}

Physics: {
  Mobility: {
    DopingDep: { Masetti: []; }
  }
  Recombination: {
    SRHRecombination = ["DopingDep"];
  }
}

Physics*Material*Silicon: {
  Mobility: {
    DopingDep: { Masetti: []; }
  }
  Recombination: {
    SRHRecombination = ["DopingDep"];
  }
  /*Traps:
  {
    MidGapTr2: {
      Type = ["Acceptor", "Gaussian", "SpatiallyGaussian"];
      Conc = 1E17; Energy = 0.0; EnSigma = 0.2;
      Center = [0.0, 0.0, 0.0]; Sigma = [0.1, 0.1, 0.1];
    }
  }*/
}

Physics*Region*RegSi1_RegSi2: {
```

```
  InterfaceTraps: {
    MidGapTr1: {
      Type = ["Acceptor", "ClipTrapsToBandgap"];
      Conc = 1E13; Energy = 0.0; EnSigma = 0.2;
    }
  }
}

Math: {
  IterationsFC = 40;
  InnerIterationsFC = 10;
  IterationsSC = 40;
  LineSearchDamping = 1.1;
  UndampedIterations = 0;
  FCSolverTolerance = 1.0;
  SCSolverTolerance = 1.;
  BankRoseDamping = 2.71;
  RelativeError = 1E-3;
  tkMultFact = 1E-46;
  tkExpMultFact = 0.25;
  deltaK = 0.1;
  AreaFactor = 1.;
  SolverSettings = [
        "InterpolateElecPotential",
        "InterpolateElecFermi",
        "InterpolateHoleFermi",
        "IncludeMobilityDensityDerivative"
        ];
}

Plot: {
  Quantities = ["ElectronDensity",
      "HoleDensity",
      "ElectricFieldY",
      "ElectricFieldX",
      "AbsElectricField",
      "ElectrostaticPotential",
      "TotalRecombination",
```

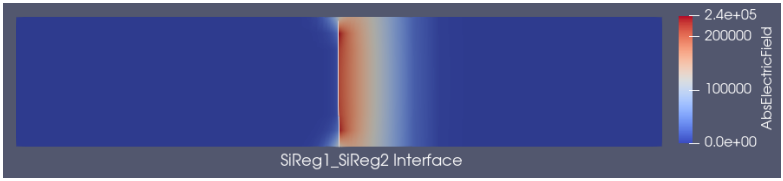
```
      "HoleMobility",
      "ElectronMobility",
      "AbsElectronCurrent",
      "AbsHoleCurrent",
      "AbsGradEFermi",
      "AbsGradHFermi"];
}

Solve:
{
  Static*PoiTemp: {
    Coupled: [ "Poisson"];
  }

  Quasistationary*RevBias:
  {
    initstep = 1E-3; minstep = 1E-7;
    maxstep = 0.1; incr = 1.35; decr = 2.;
    Ramp: {
      Voltage*Anode = -2.0;
    }
    Coupled: ["Poisson", "Electron", "Hole"]
    Plot: { Time = [0.0, 0.5, 1.]; }
  }
}
```

The above file can be simulated using the following command.

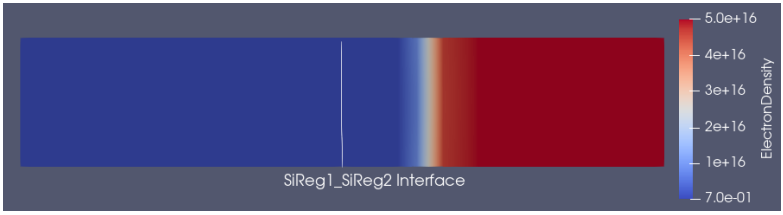
```
>> ddsolver ddsolver diode_dev.cfg
```

In the quasi-stationary ramp `RevBias`, the diode is reverse biased to -2.0V. Under reverse bias of 2.0V, acceptor traps are negatively charged creating a sheet of negative charge at `SiReg1_SiReg2` interface. Entire voltage drop appears in the n-doped region `SiReg2`. Therefore, electric field is present only in the n-doped region as seen in Fig. 10.2(a). Similarly, the depletion region appears only in n-doped region as seen from electron and hole densities in Fig. 10.2(b) and Fig. 10.2(c).

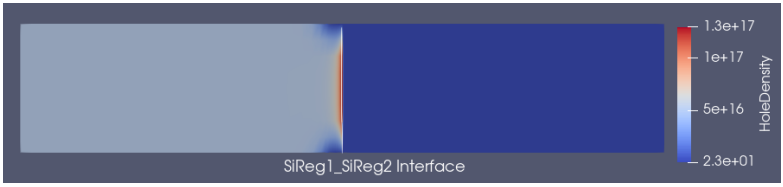
Donor traps can be defined at the interface instead of the acceptor traps by changing the following line in the above file.



(a) Electric field



(b) Electron density



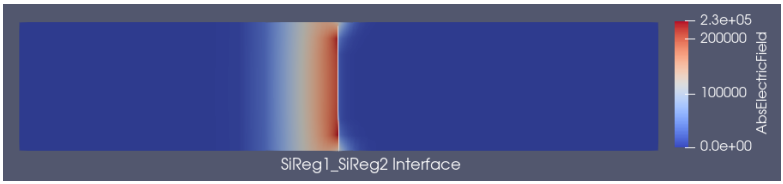
(c) Hole density

Figure 10.2: (a) Electric field, (b) Electron density, and (c) Hole density in a 2D diode structure with $10^{13}/cm^2eV$ uniform 'Acceptor' traps defined at the interface.

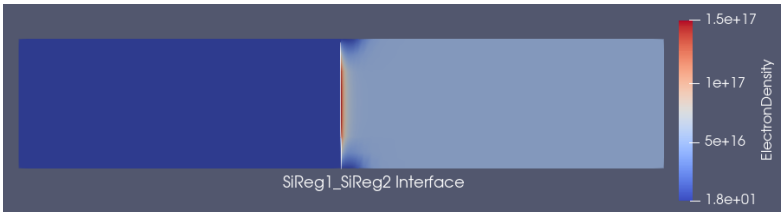
```
Type = ["Donor", "ClipTrapsToBandgap"];
```

Performing the diode simulations again yields electric field only in the p-doped region as shown in Fig. 10.3(a). Under the reverse bias, the donor traps are positively charged, creating a sheet of positive charge at `SiReg1_SiReg2` interface. Similarly, depletion appears only in p-doped region as seen from electron and hole densities in Fig. 10.3(b) and Fig. 10.3(c), respectively.

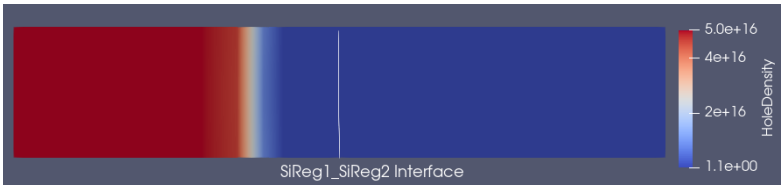
Alternately, bulk traps can be defined instead of interface traps by commenting out interface trap section, and uncommenting the `Traps` section in `Physics*Material*Silicon:`. This section defines bulk traps localized at $\vec{r} = [0., 0., 0.]\mu$m with $\sigma = 0.1\mu$m.



(a) Electric field



(b) Electron density



(c) Hole density

Figure 10.3: (a) Electric field, (b) Electron density, and (c) Hole density in a 2D diode structure with $10^{13}/cm^2eV$ uniform 'Donor' traps defined at the interface.

Chapter 11

Nonlocal tunneling models

Tunneling of electrons and holes across across a potential barrier is modeled in the DD solver by various non-local tunneling models. These models are applied on a special non-local mesh defined on the device structure.

11.1 Non-local mesh

A Nonlocal Mesh (NLM) is a set of straight lines which begin at a specified list of region-interfaces or material-interfaces and extends along the specified direction. The nonlocal mesh is defined only in the math section (`Math: { ... }`) in the DD solver config file.

A nonlocal mesh is defined in the DD solver config file as follows.

```
Math:
{
    NLLine*Line1:
    {
        RegionInterfaces = ["Reg1/Reg2", "Reg3/Reg4"]
        Length = 1; // in micrometers
        ReverseLength = 0.5; // in micrometers
```

```
        Spacing = 1E-3; // in micrometers
        Direction = [1,0,0]; // unitless
    }

    NLLine*Line2:
    {
     ...
    }
}
```

The above script defines a NLM named `Line1`. The mesh is a set of nonlocal lines starting at the region-interfaces listed in the argument `RegionInterfaces`. The lines extend along the direction given by the argument `Direction` by length given by `Length`. Additionally, the lines are also extended along the opposite direction by length specified by `ReverseLength` from the region-interface. Each of these lines is discretized with the discretization length given by `Spacing`. Multiple NLM definitions can be listed in `Math` section as given in the script.

Various non-local models utilize the NLMs defined in `Math` section. Names of the NLMs on which the models are active, are listed along with the models. The nonlocal models are listed in the next sections.

11.2 Intra-band tunneling

When the n-channel MOSFET is in off-state, electrostatic barrier under the gate prevents flow of electrons from source to drain. When the source-drain is biased, high energy electrons flow over the barrier generating a tiny thermionic leakage current. As MOSFET channel length has shrunk below few 10s of nanometers, a new leakage mechanism has emerged. Electrostatic barrier under the channel has also shortened due to the shortening of the channel length. As a result, electrons tunnel from source-to-drain creating a larger leakage current. Since electrons tunnel from the CB in the source region to the CB in the drain region, this tunneling mechanism is classified as 'intra-band tunneling' meaning within the same band. Note, that the above description is also applicable to a p-channel MOSFET when electrons are replaced with holes.

Tunneling of electrons or holes using the nonlocal tunneling model is activated in the device using the following text in the config file.

```
Physics:
{
    eNonlocalTunneling*T1: ["Lines1"]
    hNonlocalTunneling*T2: ["Lines2"]
}
```

Notice, that the models are defined in global `Physics` section instead of 'region-wise' physics sections. `eNonlocalTunneling` keyword defines nonlocal intra-band tunneling model for electron tunneling, whereas `hNonlocalTunneling` keyword defines it for hole tunneling. Names of NLMs along which the tunneling models are active, are listed along with the model definitions. These NLMs must be defined in `Math` section as described in Section-11.1.

Tunnel rate of electrons or holes is calculated along each of the nonlocal lines in the specified NLMs as follows.

Tunneling rate is calculated at every discretized point at a distance l along the nonlocal line by extending a tunnel path at tunnel electron energy $\epsilon = E_{\rm C}(l)$. The tunnel path ends at the *first point* along the line at a distance $u > l$, at which $\epsilon < E_{\rm C}(u)$. The recombination rate at l and generation rate at u due to tunneling along this tunnel path are calculated using Eq. 11.1.

$$R_{\rm t}(l, \epsilon = E_{\rm C}(l)) = \frac{|\vec{F} \cdot \hat{l}|}{72\hbar} \cdot \Big(\int_l^u \frac{dx}{\kappa(x,\epsilon)}\Big)^{-1} \cdot \exp\Big(-\int_l^u \kappa_c(x,\epsilon)dx\Big) \cdot \Big(f\big(\frac{\epsilon - F_{\rm n}(u)}{kT}\big) - f\big(\frac{\epsilon - F_{\rm n}(l)}{kT}\big)\Big) \tag{11.1}$$

$$G_{\rm t}(u) = R_{\rm t}(l, \epsilon = E_{\rm C}(l)) \tag{11.2}$$

Here, $\kappa_c(x,\epsilon) = \frac{\sqrt{2 \cdot m_e \cdot (E_{\rm C}(x) - \epsilon)}}{\hbar}$ is an imaginary κ at each x along the tunnel path. $|\vec{F} \cdot \hat{l}|$ is electric field at l along the tunnel path. $f(x) = \frac{1}{1+exp(x)}$ is Fermi-Dirac distribution function. The above equation is obtained by deriving the tunnel rate as described in the appendix of [1].

Thus, the intra-band electron tunnel rate is introduced in DD equation as a recombination rate at l and a generation rate at u. If tunneling takes place in the opposite direction (i.e. from u to l), the rate will be negative. This would correspond to electrons tunneling from u to l.

Similar expression is derived for intra-band tunneling of holes along each of the nonlocal tunnel lines.

11.3 Band-to-band tunneling

In the presence of high electric field, electrons tunnel from VB to CB resulting in generation of holes in the VB and generation of equal number of electron in the CB. This process is inherently a nonlocal process, since the location of electron generation is spatially separated from that of hole generation. In the presence of 'uniform' electric field, a local e/h generation rate can be derived as described in Chapter 9 Section 9.2. Accurate calculation of tunneling probability in the presence of non-uniform field is performed by using the nonlocal band-to-band tunneling model.

Tunneling of electrons or holes using the nonlocal tunneling model is activated in the device using the following text in the config file.

```
Physics:
{
    B2BNonlocalTunneling*T1: ["Lines1","Lines2"]
}
```

The models are defined in global `Physics` section. `B2BNonlocalTunneling` keyword defines nonlocal band-to-band tunneling model. Names of NLMs along which the given tunneling model is active, are listed along with the model definitions. These NLMs must be defined in `Math` section as described in Section-11.1.

The band-to-band tunnel rate is calculated along each of the nonlocal lines in the specified NLMs as follows.

Tunneling rate is calculated at every discretized point at a distance l along the nonlocal line by extending a tunnel path at the tunnel electron energy. If electric field at l in the direction of tunnel path is positive, then CB energy is set as tunnel energy ($\epsilon = E_C(l)$). Such

a tunnel path ends at the *first point* along the line (at $u > l$), at which $\epsilon < E_{\rm V}(u)$. Alternately, if electric field at l along the path is negative, then VB energy is set as tunnel energy ($\epsilon = E_{\rm V}(l)$). Such a tunnel path ends at the *first point* along the line (at $u > l$), at which $\epsilon > E_{\rm C}(u)$.

The tunnel rate along the tunnel path CB →VB is calculated using Eq. 11.5.

$$R_{\rm t}(l, \epsilon = E_{\rm C}(l)) = \frac{|\vec{F} \cdot \hat{l}|}{72\hbar} \cdot \left(\int_l^u \frac{dx}{\kappa(x)}\right)^{-1} \cdot \exp\left(-\int_l^u \kappa_r(x)dx\right) \cdot \left(f\left(\frac{E_{\rm V}(u) - F_{\rm n}(u)}{kT}\right) - f\left(\frac{E_{\rm C}(l) - F_{\rm n}(l)}{kT}\right)\right) \quad (11.3)$$

Here, $\kappa_r(x, \epsilon) = \frac{\kappa_{\rm c} \cdot \kappa_{\rm v}}{\sqrt{\kappa_{\rm c}^2 + \kappa_{\rm v}^2}}$ is a 'two-band' imaginary dispersion relation smoothly interpolated from CB to VB. Each of the $\kappa_{\rm c/v}$ are given by $\kappa_{\rm c/v}(x, \epsilon) = \frac{\sqrt{2 \cdot m_e \cdot (E_{\rm C}(x) - \epsilon)}}{\hbar}$.

Tunnel rate of an electron from CB to VB, calculated by Eq. 11.5, is added to the hole recombination rate at the point u and to the electron recombination rate at the point l.

Similarly, the tunnel rate along the path from VB →CB is calculated. In this case, $\epsilon = E_{\rm V}(l)$ and the occupancy factors are adjusted accordingly. It is added to the electron recombination rate at the point u and to the hole recombination rate at the point l.

11.4 Trap-to-band tunneling

Defects at the semiconductor interfaces give rise to recombination centers (also called traps). These traps capture or emit an electron to the CB and capture or emit a hole to the VB. Modeling technique of local e/h capture/emission processes has been described in Chapter 10. In addition to these processes, electrons or holes can also undergo tunneling, respectively from CB or VB, to and from the trap.

Tunneling to the trap is classified as e/h capture process, while tunneling from the trap is classified as e/h emission. Tunneling can be activated in the interface trap definition in the config file as follows.

```
Physics*Material*Silicon: {
```

```
  InterfaceTraps: {
    MidGapTr1: {
      Type = ["Donor", "Gaussian"];
      Conc = 1E17; Energy = 0.0; EnSigma = 0.2;
      eNLPs = ["Line1"];
      hNLPs = ["Line2"]
    }
  }
}
```

In the above interface trap definition, the keyword `eNLPs` activates tunneling along the NLMs specified as a list of arguments along with that keyword. Similarly, the keyword `hNLPs` activates tunneling along the NLMs listed along with that keyword.

Capture cross-section of electrons due to the tunneling process is be calculated along the nonlocal lines in the specified NLMs as follows.

At each interface vertex, energy levels of the interface traps are discretized in the band gap of the semiconductor. Electron capture cross-section at every trap energy level ($E_{\rm t}$) is obtained by calculating tunnel rate along the non-local line starting from that vertex, ending at a point u at which $E_{\rm t} > E_{\rm C}(u)$. The capture cross-section is given by.

$$\sigma_{\rm n}(\epsilon = E_{\rm t}) = 2V_{\rm T}\frac{\sqrt{8m_{\rm c}}}{\hbar^4} \cdot (E_C(u) - \epsilon)^2\sqrt{\epsilon - E_V(l)} \cdot \exp\,(-\int_0^u \kappa_c(x)dx) \cdot f(\frac{E_{\rm C}(l) - F_{\rm n}(l)}{kT}) \tag{11.4}$$

This capture cross-section is added to the capture cross-section originating from the local processes to calculate trap occupancy (see Eq. 10.1). Electron capture rate at the given trap energy level ($E_{\rm t}$) by tunneling is obtained by considering the trap density and occupancy. Similarly, electron emission rate is calculated. Net electron capture rate (capture rate − emission rate) is added to electron recombination rate at the point u. In this way, nonlocal trapping of electrons is included in the DD simulations.

Nonlocal capture and emission of holes by tunneling is modeled in the similar way as follows. Hole capture cross-section at every discretized trap energy level ($E_{\rm t}$) is obtained by calculating tunnel

rate along the non-local line starting from that vertex, ending at a point u at which $E_{\mathrm{t}} < E_{\mathrm{V}}(u)$. The capture cross-section is given by.

$$\sigma_{\mathrm{p}}(\epsilon = E_{\mathrm{t}}) = 2V_{\mathrm{T}} \frac{\sqrt{8m_{\mathrm{v}}}}{\hbar^4} \cdot (E_V(u) - \epsilon)^2 \sqrt{\epsilon - E_C(l)} \cdot \exp\left(-\int_0^u \kappa_v(x)dx\right) \cdot f\left(\frac{E_{\mathrm{V}}(l) - F_{\mathrm{n}}(l)}{kT}\right) \tag{11.5}$$

Similar to the electron-case, hole capture and emission cross-sections are used to calculate net hole capture rate, which is added to hole recombination rate at the point u.

11.5 Parameters

The above equations require electron and hole effective mass values for each of the material along the tunnel path. The effective mass parameters are defined in `NonlocalTunneling` section of the `BandStructure` part of the material config file. The electron and hole effective masses are specified with the keyword `mc` and `mv`, respectively.

11.6 Visualization

Total electron and hole tunnel rates arising from *all* the nonlocal tunneling processes can be visualized using the keywords `eNonlocalRecombination` and `jNonlocalRecombination`, respectively, in the `Plot` section in `Quantities` list.

Note: Derivatives of the nonlocal recombination processes are not added to the 'Jacobian' in the DD simulations. As a result, AC simulations *do not* consider the contributions of nonlocal tunneling processes. Additionally, convergence issues might be encountered in DC or transient simulations which include nonlocal tunneling processes, especially for high tunnel rates.

Chapter 12

Electro-thermal simulations

During device operation, carrier transport and generation/recombination processes generate heat in the device. The generated heat increases temperature locally creating temperature gradients. This results in heat transport to the heat-sinks attached to the device. Increased Temperature also alters semiconductor and metallic material properties locally, which in turn affects heat generation rate. Accurate modeling of the device characteristics necessitates solving heat transport equation together with the carrier transport equation. The DD simulator enables coupled as well as self-consistent solver for simulating carrier transport and heat transport.

12.1 Config file

An example config file which enables electro-thermal transport is provided below.

```
File: {
  Device = "diode_str.cfg";
  Out = "Diode";
  Simulation = "DD";
```

```
}

Contacts: {
  Anode: {Voltage = 0.0; Type = ["Semiconductor"]; Temperature=300.}
  Cathode: {Voltage = 0.0; Type = ["Semiconductor"];  Temperature=310

}

Physics: {
  Mobility: {
    DopingDep: { Masetti: []; }
  }
  Recombination: {
    SRHRecombination = ["DopingDep"];
  }
  ThermalProperties: {
    HeatCapacity: ["TempDep"];
    HeatConductivity: ["TempDep"];
  }
}

Math: {
  IterationsFC = 40;
  InnerIterationsFC = 10;
  FCSolverTolerance = 1.0;
  SolverSettings = ["InterpolateElecPotential",
                    "InterpolateElecFermi",
                    "InterpolateHoleFermi"];
}

Plot: {
  Quantities = ["ElectrostaticPotential",
              "TotalRecombination",
              "AbsElectronCurrent",
              "AbsHoleCurrent",
              "eJouleHeat",
              "hJouleHeat",
              "HeatGeneration",
```

```
                "Temperature",
                "ConductionBandEdge",
                "ValenceBandEdge"];
}

Solve: {
  Static*Temp: {
    Coupled: [ "Temperature"];
  }

  Static*PoiTemp: {
    Coupled: [ "Poisson", "Temperature"];
    Plot: { Time = [0.0]; }
  }

  Quasistationary*AnodeRamp: {
    initstep = 1E-3; minstep = 1E-7; maxstep = 0.1; incr = 1.35; decr
    Ramp: {
      Voltage*Anode = 2.0;
    }
    Coupled: ["Poisson", "Electron", "Hole", "Temperature"]
    Plot: { Time = [0.0, 0.5, 1.]; }
  }
}
```

The above config file has the same structure as the one presented in Chapter 2, except a few additional keywords and arguments described below.

12.2 Contacts section

Setting `Temperature` argument in the contact definitions internally creates a heat-sink with the specified temperature. For example, adding `Temperature` in the contact definition of both `Anode` and `Cathode` creates two thermal contacts with the same names.

12.3 ThermalContacts section

If a `ThermalContact` named `ThContA` is explicitly defined in the structure config file of the device, its temperature can be set by creating a separate `ThermalContacts` section and adding the thermal contact definition to it as follows.

```
ThermalContacts: {
  ThContA: { Temperature=300.; }
}
```

12.4 Physics section

`ThermalProperties` subsection can be added to the regionwise/materialwise/global `Physics` section. Active models in the region can be defined in the section. In the above file, temperature dependent heat capacity and heat conductivity have been defined by adding `TempDep` to the models `HeatCapacity` and `HeatConductivity`.

12.5 Solve section

The solver is notified to solve heat transport equation coupled with Poisson and continuity equations by adding `Temperature` keyword in `Coupled` statement as shown in the solve section.

12.6 Material config file

Parameters associated with temperature dependence of the heat capacity and the heat conductivity are set in `ThermalProperties` part of the material config file. The parameters can be defined as mole-fraction dependent by using the same settings as described in Chapter 2.

12.7 Heat transport equation

Heat generated in the active regions of the device dissipates according to the following heat transport equation.

$$c \cdot \frac{\partial T}{\partial t} - \nabla \cdot (\kappa \nabla T) = G_{\text{Heat}} \tag{12.1}$$

Here, c is the heat capacity of the material, κ is the thermal conductivity, and T is local temperature in the device. G_{Heat} is the rate of heat generation due to carrier transport as well as carrier recombination mechanisms.

12.8 Heat conductivity

Heat conductivity κ determines how fast generated heat is spread across the material. In DD simulator, temperature dependent heat conductivity has been modeled as follows.

$$\kappa = \kappa_{\text{A}} + \kappa_{\text{B}} \cdot T + \kappa_{\text{C}} \cdot T^2 \tag{12.2}$$

Here, κ_{A}, κ_{B}, and κ_{C} are the model parameters. They can be set in `TempDepConductivity` section of `ThermalProperties` part in the material config file.

12.9 Heat capacity

Heat capacity c_v determines how much heat is needed to increase temperature of a unit mass of the material by a unit temperature. In DD simulator, temperature dependent heat capacity is modeled using the following equation.

$$c_v = c_{va} + c_{vb}T + c_{vc}T^2 + c_{vd}T^3 \tag{12.3}$$

Here, c_{va}, c_{vb}, c_{vc}, and c_{vd} are the model parameters. They can be set in `TempDepCapacity` section of `ThermalProperties` part in the material config file.

12.10 Heat generation in metals

Dissipative electron transport in metals generates heat locally due to Ohmic resistance. Electron current in the metals can be expressed as $\vec{J}_{\mathrm{m}} = \sigma \nabla \phi_{\mathrm{m}}$. Heat generation rate due to the electron transport is given by the following equation.

$$G_{\mathrm{Heat,M}} = \sigma |\nabla \phi_{\mathrm{m}}|^2 \tag{12.4}$$

Here, σ is the electron conductivity of the given metal and ϕ_{m} is electron Fermi level. It is implicitly assumed that no electron generation takes place in the metal ($\nabla \cdot \vec{J}_{\mathrm{m}} = 0$) in deriving Eq. 12.4.

12.11 Heat generation in semiconductors

Dissipative electron and hole transport in semiconductors generates heat locally. Also, recombination of electrons and holes releases heat equal to the difference between electron and hole Fermi levels. Heat generation due to these contributions is given by the following equation.

$$G_{\mathrm{Heat,S}} = qn\mu_n |\nabla \phi_n|^2 + qp\mu_p |\nabla \phi_p|^2 + q(\phi_p - \phi_n)R \tag{12.5}$$

Here, $\mu_{n/p}$ are electron and hole mobilities, n and p are electron and hole densities, $\phi_{n/p}$ are electron and hole Fermi energies, and R is carrier recombination rate. First two terms on the Right Hand Side (RHS) of Eq. 12.5 correspond to heat generated during electron and hole transport. Last term of Eq. 12.5 corresponds to energy released during recombination process.

12.12 Electro-thermal simulations of Metal-Insulator system

Printed circuit boards (excluding the IC chip) are an example of a metal-insulator system. These system need to be simulated to understand the regions with high heat generation during operations. In the example below, a metal strip without any insulator or semiconductor region is simulated using the solver. The structure generation config

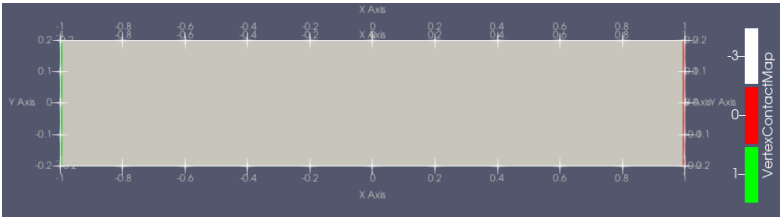


Figure 12.1: Dimensions of the simulated metal strip together with the contact information. Values of '0' and '1' correspond to the contact vertices. Value of '-3' at the vertices implies that the vertices are of the type - 'Metal'.

file and the DD solver config file can be found in the tutorials which come with the DD Solver software distribution. The structure can be generated as follows.

```
>> ddsolver str metalStrip_str.cfg
```

The structure can be viewed in paraview as follows.

```
>> paraview MetalStrip_str.xdmf
```

As shown in Fig. 12.1 the metal strip is 2 μm long and 0.4 μm wide 2D structure. The structure is always assumed to be 1μm long in Z- direction. It has contacts on the left and the right side which are also connected to the heat sinks at room temperature. The contacts are connected to the voltage source which is `Quasistationarily` ramped from 0V to 1V. This generates heat (Eq. 12.4) which raises metal temperature and increases its resistance which in turn limits the current supply. Both the contacts are connected to the heat sink at room temperature which draw out heat from the metal strip. Internally, heat conduction follows Eq. 12.1, thus creating a nonuniform temperature distribution throughout the strip. Steady-state temperature distribution at 0.5V is shown in Fig. **??**. This results in lower electron conductivity at the center of the strip as shown in Fig. **??**.

,ê,

,ê,,ê,

Appendix A

Notation and Acronyms

Acronyms

AC	Alternating Current
BC	Boundary Condition
BE	Backward Euler
BPM	Beam Propagation Method
BTBT	Band-to-band Tunneling
CB	Conduction Band
DC	Direct Current
DD	Drift-diffusion
DOS	Density of States
FDTD	Finite Difference Time Domain
HH	Heavy Hole

LH	Light Hole
MOSFET	Metal-Oxide Semiconductor Field Effect Transistor
NLM	Nonlocal Mesh
PBC	Periodic Boundary Condition
PDE	Partial Differential Equation
RHS	Right Hand Side
SRH	Schockley-Read-Hall
VB	Valence Band

Bibliography

[1] H. Carrillo-Nuñez, A. Ziegler, M. Luisier, and A. Schenk, "Modeling direct band-to-band tunneling: From bulk to quantum-confined semiconductor devices," *Journal of Applied Physics*, vol. 117, no. 23, p. 234501, 06 2015. [Online]. Available: https://doi.org/10.1063/1.4922427