Drift-diffusion Solver

Informative session

Saurabh Sant, Dr. sc. ETH

saurabh.sant@semivi.ch

semivi.ch

SemiVi LLC

Zollikon, Switzerland.

November 2, 2025

Overview

Table of Contents

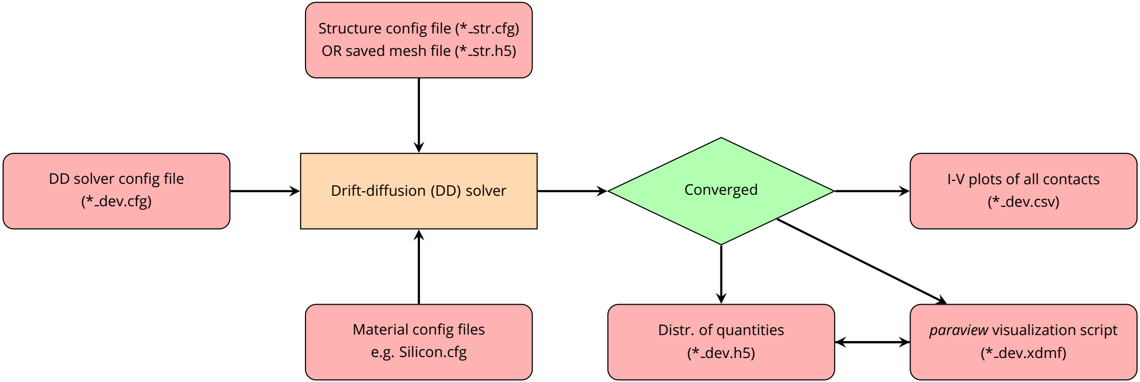
Salient features - Drift-diffusion solver

The drift-diffusion solver (DD-solver) simulates the electronic device by solving *Poisson*, and electron/hole *continuity equations*.
Salient features of the DD-solver are –

- Can perform a quasi-stationary DC 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 mixed-mode-system consisting of the semiconductor devices and the external components can be solved coupled with drift-diffusion equations.
- Physics-based models can be activated to model mobility, recombination phenomena, traps, etc. in semiconductors.
- The model parameters can be edited in the material config file, thus allowing the models to be calibrated to the experimental data.
- Convinient 'Python' interface for simulation setup and post-processing.

Table of Contents

DD solver interface and config files



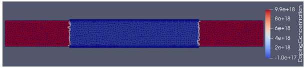
To run a DD solver simulation...

- Create and save *solver config file*, *structure config or mesh file*, *material files* (if modified), and other required files in the same folder.
- Run simulations using the following command –

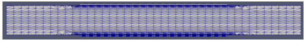
```
>> ddsolver ddsolver diode_dev.cfg
```

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

Physics*Region*RegSi: {
  Mobility: {
    DopingDep: {
      Masetti: [];
    }
    FieldDep: {
      Lombardi: [];
    }
  }
}

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

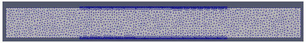
(a) Contacts



(b) Doping



(c) Mesh



(d) Mesh

Figure: Device structure

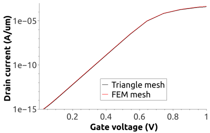
- `File`: set `Device` structure with `.cfg` file or `.h5` file.
- `Contacts`: Set properties of contacts defined in `Device`.
- `Physics`: Set *globally* active models.
- `Physics*Region*RegSi`: Set *region-wise* active models.
- `Plot`: Datasets to be stored at each specified bias point.

Table of Contents

Quasi-stationary Analysis

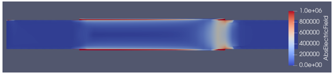
```
Math: {
  IterationsFC = 40;
  InnerIterationsFC = 10;
  FCSolverTolerance = 1.;
  SolverSettings = [
            "InterpolateElecPotential",
            "InterpolateElecFermi"
            ,"InterpolateHoleFermi"
            ];
}

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

  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];
    Math: {
      IterationsFC = 40;
    }
  }
}
```

- Math: Specify various numeric parameters and solver settings.
- Solve: Provide information on solve steps to perform *consecutively*.



(a) Transfer characteristics

(b) Electron current Vg=1V



(c) Electric field Vg=1V

(d) Electron mobility Vg=1V

Figure: Simulated device transfer characteristics and spatial distributions.

- Quasistationary, Static, and Transient ramps can be initialized.
- In Quasistationary ramp, a fictitious time variable is ramped from 0 to 1.
- Voltage*drain: Voltage of drain contact is ramped to the specified value.
- Coupled: Which equations to solved in the coupled solver. Alternately SelfConsistent solving is possible.
- Plot: Spatial distribution of Quantities is stored in an hdf file at each time-point Time.

```
Plot: {
  ACQuantities = ["ElectronDensity",
    "AbsElectricField",
    "ElectrostaticPotential"];
}

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

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

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

(a) Effect of small-signal voltage at drain

(b) Effect of small-signal voltage at gate

(c) Effect of small-signal voltage at source

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

- `ACAnalysis` specifies DC bias point(s) with `Time` at which analysis is done.
- `Nodes`: Contacts which will be included in the AC analysis.
- `ACQuantities` in `Plot` section list the quantities to be stored in AC analysis.
- `PlotFrequency`: at these frequencies the AC quantities are stored.

Mixed-mode analysis

```
Device*MOS1: {
  File: { ... }

  Contacts: { ... }

  Physics: { ... }
}

Math: {
 ...
}

System: {
  //SpiceCircuitFiles = [ "SpCkt1.cir"];
  SpiceCircuit:
    ".subckt Amplifier Vc Vg Vout
    R1 Vc Vout 1K.
    R2 Vs  0    10.
    R3 Vg  0    100.
    R4 Vc Vg    4K.
    W1 Vout Vg Vs MOS1
    .ends

    Vsupply Vc 0 DC 0.0
    X1 Vout1 Vin Vc Amplifier
    .end"
}

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

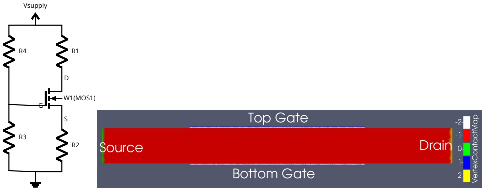


Figure: MOSFET system shown in the System section on the LHS.

`System` command solves the mixed-mode system containing electronic device which is connected to the external components.

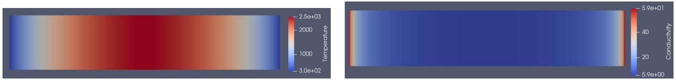
Note: Use the following for mixed-mode simulations.
`>> DDSolver systemsolver FinFETSys_dev.cfg`

- `System` section defines external circuit and links the `Device` named `MOS1` to it as follows – `W1 Vout Vg Vs MOS1`
- Solve coupled circuit, Poisson, continuity equations using – `Coupled: ["Poisson", "Electron", "Circuit"];`

Electro-thermal simulations

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

ThermalContacts: {  ... }

Physics: {
  ...
  ThermalProperties: {
    HeatCapacity: ["TempDep"];
    HeatConductivity: ["TempDep"];
  }
}

Plot: {
  Quantities = ["eJouleHeat",
                "hJouleHeat",
                "HeatGeneration",
                "Temperature"];
}

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

  Static*Poisson: {
    Coupled: ["Poisson", "Electron", "Hole",
              "Temperature"];
    Plot: { Time = [0.]; }
  }
  ...
}
```

Electro-thermal simulations solve Poisson and/or continuity equations with heat transport equation.



(a) Temperature distr. in Copper plate at 1V (b) Copper plate conductivity at 1V

Figure: Heat generation in a 2D copper plate

- `Contacts`: Each electrical contact acts as a heat-sink if `Temperature` is specified.
- If `ThermalContacts` are defined in structure config file, then list them in `ThermalContacts: { ... }`.
- In `Physics` section, `ThermalProperties` list which thermal models are active.
- In `Coupled` command, add `Temperature` for adding heat equation to the coupled solver.

Table of Contents

Solver equations

- The solver supports solving *Poisson*, *electron/hole continuity*, *circuit*, and *heat-flow* equations.
- *Poisson* equation is given by –

$$-\nabla^2\Psi(\vec{r}) = \rho_C(\vec{r}) + n(\Psi(\vec{r})) - p(\Psi(\vec{r})) \quad (1)$$

Here, $\rho_C(\vec{r})$ includes dopants, bulk, and interface charges, and charged traps.

- *Electron/hole continuity* equations are given by –

$$\frac{\partial n}{\partial t} = \nabla\cdot\vec{J}_n/q - (R_n - G_n) \text{ and } \frac{\partial p}{\partial t} = -\nabla\cdot\vec{J}_p/q - (R_p - G_p) \quad (2)$$

Here, $R_{n/p} - G_{n/p}$ term include all the recombination mechanisms, e.g. SRH, Band-to-band, impact ionization, recombination at bulk/interface defects.

- *Heat-flow* equation is given by –

$$c\cdot\frac{\partial T}{\partial t} - \nabla\cdot(\kappa\nabla T) = G_{\text{Heat}} \quad (3)$$

Here, G_{Heat} includes heat generation in semiconductors as well as metals.

- *Circuit equation* is solved by *modified nodal analysis*.

Nonlinear solver: The DD solver uses a damped Newton's solver reported in Bank et al.

$$\mathbf{g}'(\vec{x}_i)\cdot\vec{dx}_i = -g(\vec{x}_i) \quad (4)$$

$$\vec{x}_{i+1} = \vec{x}_i + \gamma\vec{dx}_i \quad (5)$$

Damping parameter (γ) calculated such that $\frac{\|g(\vec{x}_{i+1})\|}{\|g(\vec{x}_i)\|} < 1$ and $\frac{\|g(\vec{x}_{i+1})\|}{\|g(\vec{x}_i)\|} \to 1$.

Coupled solver: All equations are coupled to calculate the Jacobian ($\mathbf{g}'(\vec{x})$).

$$\mathbf{g}'(\vec{x}) = \begin{bmatrix} \frac{\partial P}{\partial\psi} & \frac{\partial P}{\partial n} & \frac{\partial P}{\partial p} \\ \frac{\partial J_n}{\partial\psi} & \frac{\partial J_n}{\partial n} & \frac{\partial J_n}{\partial p} \\ \frac{\partial J_p}{\partial\psi} & \frac{\partial J_p}{\partial n} & \frac{\partial J_p}{\partial p} \end{bmatrix} \quad (6)$$

Self-consistent solver: Equations solved one after the other until convergence is reached.

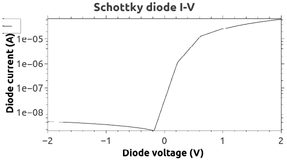
$$\mathbf{g}'(\vec{x}) = \frac{\partial P}{\partial\psi} \text{ then } \mathbf{g}'(\vec{x}) = \frac{\partial J_n}{\partial n} \text{ then } \mathbf{g}'(\vec{x}) = \frac{\partial J_p}{\partial p}$$

Contact boundary conditions

Contact properties are assigned in the *Device* config file (e.g. diode_dev.cfg) –

```
Contacts: {
  Anode: {
    Voltage = 0.0; Type = ["Semiconductor"];
    SchottkyBarrier = 0.5; Resistance=1E1;
  }
  Cathode: {
    Voltage = 0.0; Type = ["Semiconductor"];
    Resistance=1E1;
  }
}
```

(a) Electrostatic potential under reverse bias



(b) I-V plot of the Schottky diode

Figure: Heat generation in a 2D copper plate

Default boundary conditions for the drift-diffusion equations –

- Electric field normal to the device-domain boundary is zero – $\nabla\psi \cdot \hat{n} = 0$.
- Electron current (J_n) and Hole current (J_p) at the semiconductor-insulator/gas boundary is zero, – $J_n \cdot \hat{n} = J_p \cdot \hat{n} = 0$.
- Note: The above equations impose *reflective BCs* at the domain boundaries.

Semiconductor contact boundary conditions –

- Two types of semiconductor contacts – *Ohmic* and *Schottky* contacts.
- Semiconductor contacts are, by default, *Ohmic* contacts.
- *Schottky* contact is set by adding an argument `SchottkyBarrier=0.5` in the contact definition. Here 0.5eV is a Schottky barrier. The barrier enters the electrostatics as follows.
- A resistive contact is set by adding an argument `Resistance=0.1.` in the contact definition. Here, 0.1 Ω is the contact resistance.

Insulator contact boundary conditions –

- Contacts at the insulator boundary are used to specify metal/insulator interface of the gate.

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

Band-structure models

Physical models are activated in the *Device* config file (e.g. diode_dev.cfg) –

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

Physical model parameters are set in *Material* parameter file (e.g. Silicon.cfg) –

```
BandStructure: {
  BandGap: {
    Chi0 = 4.05;  Eg0 = 1.1;
    alpha = 1E-4; beta = 270.0;
    ...
  }

  OldSlotBoom: {
    Nref = 1E20; Eref = 0;
  }
  ...

  DoS: {
    Formula = 0;
    Nc300 = 2E19; Nv300 = 2E19;
    me = 0.91; mh = 0.53;
  }
}
```

- Temperature dependence of the band gap is modeled by Varshneys law.

$$E_g(T) = E_g(T_0) + \frac{\alpha T_0^2}{T_0 + \beta} - \frac{\alpha T^2}{T + \beta} - E_{bgn} \tag{7}$$

- Band-gap narrowing (E_{bgn}) caused by high dopant conc. can be modeled by *one of the* following models – `DelAlamo`, `Slotboom`, or `BennetWilson` models.
- Relation between Fermi energy and carrier density is given by Fermi-Dirac integral.

$$n(\vec{r}) = N_C \mathcal{F}_{1/2}\left(\frac{q(\phi_n - E_C)}{kT}\right) \text{where} \mathcal{F}_{1/2}(\eta) = \int_{\epsilon=0}^{\infty} \frac{\epsilon^{1/2}}{1 + \exp(\epsilon - \eta)} d\epsilon \tag{8}$$

Analytic approximation of Fermi integral is modeled by –

- $\mathcal{F}_{1/2}(\eta) = \exp(\eta) \cdots$ as in `Boltzmann` model.
- $\mathcal{F}_{1/2}(\eta) = \frac{1}{\exp(-\eta) + 0.27} \cdots$ as in `ApproximateFermi` model.

Note: The latter model is valid over a slightly wider range of η.

- DOS in Eq. 8 can be set from `niFixed` (user-defined intrinsic carrier density) *or* by specifying `Nc300`, `Nv300` *or* from `me`, `nv`.

Defining material grading

Physical models are activated in the *Device* config file (e.g. diode_dev.cfg) –

```
Physics*Region*Reg1: {
  MoleFraction: {
    BulkXFraction = 0.1;
    Direction = [1., 0., 0.];
    Position = [0.,0.,0.];
    EndVal = 0.;
    Distance = 0.1;
    Shape = "Quadratic";
    Bow = 0.01;
  }
}
```

Composition dependent parameters are set in *Material* parameter file (e.g. AlGaAs.cfg) –

```
BandStructure: {
  BandGap: {
    // ( [XFrac0, val0, ... )]
    Chi0*Xdep = [0., 4.07,
       0.45, 3.575, 1., 3.5];
    Eg0*Xdep = [0., 1.424,
       0.45, 1.985, 1., 2.17];
  }
}
...
Recombination: {
  SRHRecombination: {
    // e-specific x-dep params
    taumaxe*Xdep =[0.,1E-6,1.,1E-6];
    // h-specific x-dep params
    taumaxh*Xdep =[0.,1E-6,1.,1E-6];
    // standard params [e,h]
    taumin = [1E-6, 1E-6];
  }
```

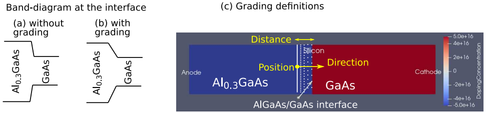


Figure: Grading at the hetero-interface

- Material mole fraction (*x*) in a region is set in `MoleFraction`.
- In the bulk of the region, *x* set by `BulkXFraction`.
- *x* changes from its bulk value to the `EndVal` starting from `Position` in the `Direction`. *x* takes a constant value at a plane normal to `Direction`.
- Shape of the 1D profile is given by `Shape`. σ of exponential and Gaussian shapes is given by `Distance`.

Mole-fraction (*x*) dependence of a parameter in a material config file.

- Band-gap dependent material parameters (e.g. `Eg0`) are listed as a pair of mole-fraction and value in brackets `(...)` instead of a single value.
- Band-gap dependent parameters of electrons and holes (e.g. `taumax`) are listed as triplets of mole-fraction, val_e, and val_h in brackets `(...)`.

Mobility models

Physical models are activated in the *Device* config file (e.g. diode_dev.cfg) –

```
Physics: {
  Mobility: {
    DopingDep: { Masetti: []; }
    FieldDep: { Lombardi: []; }
    HighFieldSat: { Canali = []; }
  }
  ...
}
```

Physical model parameters are set in *Material* parameter file (e.g. Silicon.cfg) –

```
Mobility: {
  ConstantMob: {
    mu0 = [1450.0, 450.0];
    alpha = [1.4, 1.4];
  }

  Masetti: {
    mumin1 = [52.2, 44.9];
    mu1 = [43.4, 29.];
  }

  Canali: {
    vsat0 = [1.07E7, 8.37E6];
    vsatexp = [0.87, 0.52];
  }

  Lombardi: {
    B = [4.75e7, 9.925e6];
    C = [5.8e2, 2.947e3];
  }
}
```

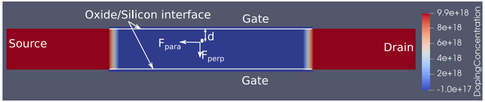


Figure: Mobility degradation at oxide/Si interface

- `ConstantMob` model for e/h mobility in low-doped Semiconductors. It is always active.
- `Masetti` model for Doping dependent e/h mobility $\mu_b(N_D)$. It depends on N_D – the net doping at the given location.
- `Lombardi` model describes mobility degradation in the MOS channel due to phonon scattering (μ_{ac}) and surface-roughness scattering (μ_{sr}). Both depend on $F_\perp$ – electric field perpendicular to oxide-Silicon interface.
- `Canali` model – mobility saturation due to high electric field along the transport direction.
- Mattheisen's rule: Low field mobility models are combined using Mattheisen's rule as follows.

$$\frac{1}{\mu_{\mathrm{low}}} = \frac{1}{\mu_0} + \frac{1}{\mu_b} + \exp\left(-\frac{d}{l_{\mathrm{crit}}}\right)\left(\frac{1}{\mu_{\mathrm{ac}}} + \frac{1}{\mu_{\mathrm{sr}}}\right) \quad (9)$$

Nonradiative Generation-Recombination models

Physical models are activated in the *Device* config file (e.g. diode_dev.cfg) –

```
Physics: {
  Recombination: {
    SRHRecombination = ["DopingDep"];
    Band2BandGen = [ ];
    AvalancheGen = ["vanOverstraeten" ];
  }
  ...
}
```

Physical model parameters are set in *Material* parameter file (e.g. Silicon.cfg) –

```
Recombination: {
  SRHRecombination: {
    taumax = [1E-6, 1E-6];
    taumin = [1E-9, 1E-9];
  }

  BandToBandGen: {
    A1p5 = 3.4E21;
    B = 21.9E6;
  }

  VanOverstraeten: {
    al = [7.03E5, 1.582E6];
    bl = [1.231E6, 2.036E6];
  }
}
```

The following models describe non-radiative G-R phenomena in Semiconductors.

- `SRHRecombination` models G-R rate arising mainly from deep energy traps. Doping-dependence of carrier life-times is activated by `DopingDep` keyword and are calculated by `Scharfetter` model.
- `Band2BandGen` model accounts for generation of carrier by band-to-band tunneling in high electric field regions. It calculates generation rate by the following expression–

$$G_{\mathrm{btb}} = A_p \cdot |F|^p \cdot \exp\left(-\frac{B}{|F|}\right) \tag{10}$$

  Here, $p = 1., 1.5, 2.0$.
- `AvalancheGen` model accounts for generation of carriers by impact-ionization process which takes place when the semiconductor is subject to very high electric field. The generation rate is given by,

$$G_{II} = \alpha_n \left|\vec{J}_n\right| + \alpha_p \left|\vec{J}_p\right| \tag{11}$$

  The multiplication factors $\alpha_{n/p}$ are modeled by `vanOverstraeten` model as follows.

$$\alpha_{n/p}(F_{\mathrm{ava,n/p}}) = \gamma \cdot a_{n/p} \cdot \exp\left(-\frac{\gamma b_{n/p}}{F_{\mathrm{ava,n/p}}}\right) \tag{12}$$

Radiative G-R models

Modify the *Device* config file (e.g. diode_dev.cfg)

```
File: {
  OpticalGenerationFile1 =
          "fdtdFinFET_TimeAvg_AvgMid_fdtd.h5";
  OpticalGenerationField = "AbsElectricField";
}
...
Physics: {
  Recombination: {
    GenerationFromFile = ["FileIds = 1",
             "Scaling = 0", "TimeRamp"];
    ApproxRadiativeRec = [ ];
    ConstantGeneration = ["Rate = 1E10",
             "TimeRamp"];
  }
}
...
Solve: {
  Quasistationary*1: {
    Ramp: {
      Voltage*gate = 1.0;
      ConstantGeneration*Rate = 1E15;
      GenerationFromFile*Scaling = 1;
    }
  }
}
```

And, the *Material* parameter file –

```
Recombination: {
  RadiativeRec: {
    C = 1E18; // units of cm^3/sec
    A0Stim = 1E18; // units of cm^3/sec
    A1Stim = 1E18; // units of cm^3/sec
    N0Stim = 1E10; // units of 1/cm^3
  }
}
```

- `ApproxRadiativeRec` analytically models G-R rate in the LEDs or LASERs. Recombination rate –

$$R = C \cdot n \cdot p \cdot \left(\frac{T}{T_{\mathrm{par}}}\right)^{\alpha} \qquad (13)$$

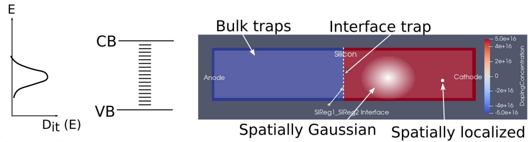
- `ConstantGeneration` model adds a constant generation rate in the regions where it is active.
- `GenerationFromFile` model adds a spatially varying generation rate (typically taken from optical simulations) to the regions where it is active. It is scaled by the `Scaling` factor before adding. Add the following arguments to the `File` section.
  - `OpticalGenerationFile1`: Specify the hdf5 file generated by the FDTD/BPM/Mode solvers. 20 such files can be specified.
  - `OpticalGenerationField`: Specify the optical field name to be added from the file.
- In last two models, generation rate can be *ramped up* in `Quasistationary` `transient` ramps to achieve convergence. Modify `Solve` command as shown in the file.

Modify the *Device* config file (e.g. diode_dev.cfg)

```
Physics*Material*Silicon: {
  Traps: {
    MidGapTr1: {
      Type = ["Donor", "Gaussian",
              "FromValenceBand"];
      Conc = 1E17; Energy = 0.0;
      EnSigma = 0.2;
    }
  }
}
...
Physics*Region*RegSi1_RegSi2: {
  InterfaceTraps: {
    MidGapTr1: {
      Type = ["Acceptor",
              "ClipTrapsToBandgap"];
      Conc = 1E13; Energy = 0.0;
      EnSigma = 0.2;
    }
  }
}
```

And, the *Material* parameter file –

```
Recombination: {
  Traps: {
    vth0 = [2.3E6, 1.7E6];
    Jfactor = [0, 0];
    Gfactor = [2, 2];
    Xsection = [1E-15, 1E-15];
    ConstEmissionRate = [0, 0];
  }
}
```



(a) Energetic trap distr. (b) $D_{it}(E)$ discretization (c) Spatial trap distribution

Figure: Trap definitions

- *Bulk traps* are instantiated in `Traps` subsection of *region* `Physics`, whereas *interface traps* in `Traps` subsection of *region-interface* `Physics` section.
- Trap types (e.g. `Donor`), concentration, and distribution (e.g. `Gaussian`, `FromValenceBand`) are specified in the trap instantiating groups.
- (De-)trapping rates are calculated by V-section or J-section models.

$$c_C^n = \sigma_n \left((1 - g_n^J) \cdot v_{\text{th}}^n \cdot n + g_n^J \cdot \frac{|\vec{J}_n|}{q} \right) \text{ and } e_C^n = \sigma_n \cdot v_{\text{th}}^n \cdot n1/g_n + e_C0^n \tag{14}$$

- Trapping parameters σ_n, v_{th}, g^J, g_n, etc. are set in the material config file.
- Trap occupancy set by the *principle of detailed balance*, $f^n = \frac{c_C^n + e_V^p}{c_C^n + c_V^p + e_C^n + e_V^p}$

Conclusions

- Drift-diffusion solver supports,
  - quasistationary and transient simulations
  - small-signal analysis,
  - system analysis,
  - electro-thermal analysis.
- Various physical models necessary for semiconductor device modeling are implemented.
- Development of new physical models continues...
- Open to collaborations with our customers / partners.

Table of Contents

Easy-to-install licenses

For ordering a license, along with the name and the organization details, please also provide –

- For the node-locked licenses: Ethernet mac address of the client machine on which the software will run.
  OR
- For the server licenses: Ethernet mac address of the server machine at the client organization.

If you purchased one or more node-locked licenses, you will receive the following license file by secured email.

1. NodeLockedLicense_<id>_<Info>.lic, where <id> stands for license id and <Info> stands for customer identification in short.

Copy the license file to `/var/local/oesoft/licenses/` on the machine whose mac-address has been provided and change its access rights to `777`.

If you purchased one or more server licenses, you will receive the following license file by secured email.

1. ServerLicense_<id>_<Info>.lic

Copy the license file to `/usr/share/oesoft/licenses/` on the *server machine* whose mac-address has been provided.

The End

Questions? Comments?