Drift-Diffusion DD

Particle transport is treated in the drift-diffusion approximation

- Particle flux is written in terms of the electro-chemical potentials, eg.

$$j_n = \mu_n n \nabla \phi_n, \quad j_p = -\mu_p p \nabla \phi_p$$

- Particle densities are modeled assuming local equilibrium, eg. electrons:

$$n = N_c F_{1/2}\left(\frac{\phi_n - E_c}{k_B T}\right)$$

- Band parameters are calculated from k·p parameterisations including strain
- For electrons/holes:

$$\left.\begin{aligned} \nabla j_n &= \nabla\left(\mu_n n \nabla \phi_n\right) = -R(n,p) \\ \nabla j_p &= \nabla\left(-\mu_p p \nabla \phi_p\right) = -R(n,p) \\ \text{+ Poisson equation} \quad & \nabla\left(\varepsilon \nabla \varphi - \underset{\uparrow}{\boxed{P}}\right) = e(n - p + N_a^- - N_d^+) \end{aligned}\right\}$$

Piezo- and pyropolarization

Module
<u>Driftdiffusion</u>

Declaration

Module driftdiffusion {
  name = *somename*
  regions = *set_of_regions*

Physics
 { *somemodel* { } }

Contact *contact_name*
{}

Section
Physics

for simulation on a strained system

for coupled electrothermal simulations

```
Physics
 {
   strain_simulation = my_strain

   thermal_simulation = my_therm

   <Physical_models>

}
```

Band parameter models

for both bands

```
band_properties # or conduction_band or valence_band
{
     regions =  ……
     density_of_states [type]
             {
                       <parameters>
             }
}
```

Band parameter models

Implemented DOS models

- ➢Bulk (default)
- ➢Bulk kp

- ➢Constant
- ➢Gaussian

- ➢Quantum

Band parameter models

Bulk DOS

band edge and dos mass for each band

```
valence_band
{
        density_of_states bulk
        {
          dos_mass = <hole_mass>
           ………
        }
}
```

Band parameter models

bulk_kp DOS:

band parameters from **bulk kp** model

including Pikus-Bir strain corrections $\longrightarrow$

```
band_properties
{
        density_of_states bulk_kp
        {
          strain_simulation = <strain>  … …
        }
}
```

Drift-Diffusion DD

Band parameter models

Gaussian DOS:

$$g(E,\sigma)dE = \frac{N_0}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\frac{(E-E_{C,V})^2}{\sigma^2}\right) dE$$

```
band_properties
{
          density_of_states gauss
          {
             N0 = <value>
            sigma = <value>
          }
}
```

Band parameter models

Quantum DOS

sim. providing quantum density

add classical density model for the barriers regions or when quantum density is not available

```
conduction_band | valence_band
 { density_of_states quantum
     {
      quantum_simulation = quantum_sim

       [barrier_regions = regions ]

          classical_DOS bulk_kp
          {
          strain_simulation = strain_sim
          }
     }
}
```

Physical Model
Mobility

Implemented mobility models

- Constant
- Doping_dependent
  - Masetti (Si)
  - Arora (GaN)
- Field-dependent
- Field assisted
- Hopping

For both particles:

```
mobility type
{
 }
```

Or:

```
electron_mobility type
{
 }

hole_mobility type
{
 }
```

Physical Model
Mobility

mobility *constant* {
mu = *mu_0* }

$$\mu_{const} = \mu_0 (T/T_0)^{-\gamma}$$

mobility *doping_dependent*{
[Masetti]
[Arora] }

mobility *field_dependent*{
low_field_model = [doping_dependent | constant]}

$$\mu = \frac{\mu_{lowfield}}{\left(1 + \left(\frac{\mu_{lowfield}|\mathbf{E}|}{v_{sat}}\right)^{\beta}\right)^{1/\beta}}$$

Physical Model
Mobility

mobility *field_enhanced* {
 mu0 = <*value*>
 E0 = <*value*>
}

$$\mu = \mu_0 e^{\sqrt{|E|/E_0}}$$

Field assisted mobility model:
enhancement of the carrier mobility by an electric field in organic semiconductors

mobility *hopping_mobility* {
 sigma = <*value*>
 N0 = <*value*> }

From Miller-Abrahams hopping model between localized states with a gaussian energy distribution

Polarization models
Polarization

Implemented models:

- Pyroelectric polarization
- Piezoelectric polarization

```
 polarization  pyro
{
  [Pz = ..]
  [P = (Px,Py,Pz)]
}
```

```
 polarization  piezo
{
  [strain_simulation = ..]
}
```

Drift-Diffusion

Physical Model

Thermoelectric

Used for self-consistent thermal/drift-diffusion simulation

Implemented models:

- Constant
- Diffusivity model

$$j_n = \mu_n n \left(\nabla \phi_n + P_n \nabla T \right)$$
$$j_p = -\mu_p p \left(\nabla \phi_p + P_p \nabla T \right)$$

thermoelectric_power *constant* {
[P_n, P_p from database] }

thermoelectric_power {
type= *diffusivity_model* }

Drift-Diffusion DD

Interface states
Trap

Implemented models:

- eNeutral
- hNeutral
- donor
- acceptor

```
 trap type
{
  [regions = list_regions or interface]
  Nt= <trap density>
  Et  =  <energy level>
  reference  = <ref. Energy>}
```

tiberlab

Drift-Diffusion DD

Interface states

Trap

eNeutral: normally neutral el.trap

$$n_t = \frac{N_t}{1+\exp(\frac{E_{trap}-E_{F,n}}{k_B T})}$$

hNeutral: normally neutral hole trap

$$p_t = \frac{N_t}{1+\exp(-\frac{E_{trap}-E_{F,p}}{k_B T})}$$

donor: normally charged el trap

$$N_t^+ = N_t - \frac{N_t}{1+\exp(\frac{E_{trap}-E_{F,n}}{k_B T})}$$

acceptor: normally charged hole trap

$$N_t^- = N_t - \frac{N_t}{1+\exp(-\frac{E_{trap}-E_{F,p}}{k_B T})}$$

Drift-Diffusion DD

Physical Models
Recombination

recombination *srh* {
  tau_n = …
  tau_p = ….. }

$$R = \frac{np - n_i^2}{\tau_p(n+n_i) + \tau_n(p+n_i)}$$

recombination *direct*{
  C = …
  }

$$R = C\left(np - n_i^2\right)$$

recombination *auger*{
  C_n = …
  C_p = …}

$$R = \left(C_n n + C_p p\right)\left(np - n_i^2\right)$$

recombination *langevin*{
  gamma = …
}

$$R_{Langevin} = \gamma\frac{e}{\varepsilon}(\mu_n + \mu_p)(np - n_0 p_0)$$

Physical Models
Tunneling

Simplified tunneling models implemented as local generation-recombination models

Trap-assisted tunneling:
Hurkx model for modified SRH

```
recombination  srh
 {
          Et = 0.56
          reference = cb

          trap_assisted_tunneling = true
}
```

Trap energy

Tunneling through the Trap is activated

Physical Models
Tunneling

Simplified tunneling models implemented as local generation-recombination models

Band-to-band tunneling:

$$G^{b2b} = BE^{\sigma}exp(E_0/E)$$

```
recombination  band2band
 {
          B = 4e14
          E0 = 1.9e7
          sigma = 2.5
}
```

Drift-Diffusion DD

Boundary conditions
Contact

Implemented models:

➢– Ohmic contact
➢– Schottky contact
➢– Interface with interface states or fixed charge density (Trap model)

```
Contact  anode{
type  = ohmic
[regions = (..)]
voltage  =  $Vd}
```

```
Contact  gate{
type  = schottky
[regions = (..)]
barrier = 3.1
voltage  =  $Vd}
```

Module ***Sweep***

Simulation executed

iteration variable (defined in Contact)

number of steps of the characteristic

```
Module sweep
{
  name = sweep_drain
  solve = driftdiffusion

  variable = $Vd
  start = 0.0
  stop = 2.0
  steps = 40

  plot_data = true
}
    {
```

Module *Sweep/2*

Simulation executed is another ***sweep*** (for nested loops)

iteration variable (defined in Contact)

number of steps of the characteristic

```
 Module sweep
{
name = sweep_gate
solve = sweep_drain

  variable = $Vg
  start = 0.0
  stop = 1.5
  steps = 6
}
}
    {
```

Calculation of Id/Vg Transfer characteristic

```
Module sweep
{
  name = sweep_drain
  solve = driftdiffusion

  variable = $Vd
  start = 0.0 # 0.2
  stop = 1.0
  steps = 5 # 4
}
```

```
 Module sweep
{
  name = sweep_gate
  solve = driftdiffusion

  variable = $Vg
  start = -0.5
  stop = 1.5
  steps = 100
 }
```

Module
Sweep/3

Sweep on **drain** to get the bias , then loop on **gate** with the fixed drain Voltage, to get transfer char.

```
Simulation
{
  temperature = 300
  solve = (sweep_drain, sweep_gate)
  resultpath = output_transchar

  output_format = vtk
}
```

Drift-Diffusion DD

Self-consistent/1
Schrödinger-Poisson

```
Physics
 { conduction_band | valence_band
     { density_of_states quantum
     {
       quantum_simulation = quantum_sim

       [barrier_regions = regions ]

            classical_DOS bulk_kp
            {
            strain_simulation = strain_sim
            }
     }
}
```

sim. providing quantum density

add classical density model for the barriers regions or when quantum density is not available

tiberlab

Self-consistent/2
Schrödinger-Poisson

Specify order of execution in a single step of the cycle

Self-consistent cycle repeated until mimimum error (*tolerance*)

```
Module selfconsistent
{
  solve = (quantum_sim, dd_sim)
  max_iterations = 10

  relative_tolerance = 1e-8
}
```