Semiconductor Device Simulation
Using DEVSIM

Juan Sanchez

July 2, 2018
Introduction

- PDE semiconductor device simulator
- Finite volume method
- Solves 1D, 2D, and 3D structures
- External meshing tools or internal mesher
- Symbolic model evaluation
- Visualization using standard output formats
Device Equations

- Drift-diffusion equations

\[ \nabla^2 \varphi = q \left( p - n + N_D - N_A \right) \]  (Poisson)

\[ \frac{\partial n}{\partial t} = \frac{1}{q} \nabla \cdot \vec{J}_n + G_n - R_n \]  (Electron Continuity)

\[ \frac{\partial p}{\partial t} = -\frac{1}{q} \nabla \cdot \vec{J}_p + G_p - R_p \]  (Hole Continuity)
Example – 1D Diode
Example – 2D MOSFET
Example – 2D MOSFET
Introduction

- Project started in 2008
- Open source since 2013 [https://devsim.org](https://devsim.org)
- C++ using STL, C++-11, and templates
- Platform Agnostic (Linux, OS X, Windows)
- Uses Python scripting to set up equations and control simulation
- Approximately 64,000 lines of code

[https://www.openhub.net/p/devsim](https://www.openhub.net/p/devsim)
Architecture – Analysis

• Nonlinear simulation
  – dc
  – transient

• Linear analysis
  – small-signal ac
  – sensitivity (impedance field)
  – noise
Architecture – Scripting

- Models implemented using scripting
  - Faster development cycle
  - Design for efficiency
- Symbolic differentiation
  - Faster development time
  - Add derivatives w.r.t. new variables
  - Common subexpression elimination
Architecture – Python

- well defined and consistent
- avoids domain specific languages with limited debugging
- provides users more control
- has numerous libraries for analysis and visualization
Architecture – Numerics

- **BLAS and LAPACK**
  - Used for dense matrix and vector operations, geometric processing
  - Optimized for most platforms
  - Called by sparse matrix factorization

- **SuperLU, MKL Pardiso** used for sparse matrix factorization

- **Iterative Math Library** used for GMRES
SYMDIFF

- Symbolic differentiation library
- Open source https://symdiff.org
- String based approach with dynamic binding of names to referred quantities
  - Constants
  - Independent variables
  - Models
SYMDIFF – Parser

- Uses rules of precedence and associativity
- Has `simplify` algorithm to reduce cost

```latex
<<<<<< diff(a + b + c^2, c)
(2 * c)
<<<<<< diff(x^x, x)
(((x * (x^(-1))) + log(x)) * (x^x))
<<<<<< simplify(diff(x^x, x))
((1 + log(x)) * (x^x))
```
Defining functions requires specification of new function and derivatives w.r.t. each named variable argument.

```lisp
> define(sqrt(x), 0.5 * x^(-0.5))
sqrt(x)
> diff(sqrt(x*y), y)
((0.5 * ((x * y)^(-0.5))) * x)
```
SYMDIFF – Models

- Models allow
  - creation of new PDEs
  - hierarchy for sub-expression elimination
  - ability to specify or generate derivatives

- Models dynamically bound by name
  - `diff(Model,x) is Model::x`
Element Assembly

- Expressions evaluated at run time
- Symbolic derivatives of models for Jacobian assembly
- Assembles bulk, interface, and contact equations
- Circuit boundary conditions
Node Models
Node Models – Shockley Read Hall

\[
U_{SRH} = \frac{np - n_i^2}{\tau_p (n + n_1) + \tau_n (p + p_1)}
\]

USRH="(Electrons*Holes - n_i^2)/ (taup*(Electrons + n1) + taun*(Holes + p1))"
Gn = "-ElectronCharge * USRH"
Gp = "+ElectronCharge * USRH"
NodeModel("USRH", USRH)
NodeModel("ElectronGeneration", Gn)
NodeModel("HoleGeneration", Gp)
for i in ("Electrons", "Holes"):
    NodeModelDerivative("USRH", USRH, i)
    NodeModelDerivative("Gn", Gn, i)
    NodeModelDerivative("Gp", Gp, i)
Node Models – Shockley Read Hall
Edge Models
Edge Models

- Electric field ($\mathcal{E}$) w.r.t potential ($\varphi$)

```python
def edge_model(device=device, region=region,
              name='\mathcal{E}',
              equation='(\varphi@n0 - \varphi@n1)\times\text{EdgeInverseLength}')
```

```python
def edge_model(device=device, region=region,
              name='\mathcal{E}:\varphi@n0',
              equation='\text{EdgeInverseLength}')
```

```python
def edge_model(device=device, region=region,
              name='\mathcal{E}:\varphi@n1',
              equation='\text{EdgeInverseLength}')
```
Element models are used to simulate mobility with respect to electric field normal and perpendicular to current flow
BJT Example

Available
https://github.com/devsim/devsim_bjt_example
BJT – DC Analysis

\[ V_{be} \text{ (V)} \]
\[ \beta, I_c, I_b \]

\[ V_{ce} \text{ (V)} \]
\[ I_c \text{ (A/cm)} \]
BJT – AC Analysis

![Graph showing BJT AC analysis results with parameters and data points related to collector current (Ic) and frequency (f) for different beta (β) values.](image)

- Collector current (Ic) in A/cm
- Frequency (f) in Hz
- Beta (β) values: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0

The graph illustrates the variation of Ic with respect to f for different β values, highlighting the significance of beta in determining the AC characteristics of the BJT.
Density Gradient

- Quantum correction method for carrier density near interfaces
- Carrier quantization effects

\[ \Lambda_e = -b_n \frac{\nabla^2 \sqrt{n}}{\sqrt{n}} \]

\[ \frac{\nabla^2 \sqrt{n}}{\sqrt{n}} = \frac{1}{2} \left\{ \nabla^2 \log n + \frac{1}{2} (\nabla \log n)^2 \right\} \]

Using \( n = \exp(u) \)

\[ \int \Lambda_e \, d\nu = -\frac{b_n}{2} \left\{ \int \nabla u \cdot d\nu + \frac{1}{2} \int (\nabla u)^2 \, d\nu \right\} + \frac{b_{n_{ox}}}{x_n} \sigma_{int} \]
Density Gradient
Density Gradient

CV Curves $t_{ox}=3$ (nm)

- $N_A=10^{17}$ (#/cm$^3$)
- DG
- $N_A=10^{18}$ (#/cm$^3$)
- DG
- $N_A=10^{19}$ (#/cm$^3$)
- DG