Symbolic Model Evaluation for TCAD Device Simulation

Juan Sanchez
Devsim LLC
P.O. Box 90636, Austin, TX 78709
Email: juan.sanchez@devsim.com

***Abstract*—Symbolic model evaluation is a powerful method for developing models for technology computer-aided design (TCAD) device simulation. It gives users the ability to accurately describe physical phenomena. Coupled with automatic creation of derivative expressions, new models can be rapidly developed with performance rivaling source code approaches. A new device simulation tool is presented with a drift-diffusion example.**

I. Introduction

Reducing model development time is an important need for TCAD [1]. It enables quicker adaptation to new modeling needs, and improves TCAD's relevance in the device development cycle.

Commercial vendors implement models in computer source code, which is compiled into a machine readable format. While efficient, it limits user configuration to a few parameters. In addition, new model availability is based on the vendor's schedule, and prioritized based on their resources.

When given a means to create new models, users are required to program in a computer language, like C++ [2], [3], and implement model derivatives. They are often limited to specific models, such as recombination terms in the device equations, and be aware of model dependencies assumed by the simulator [4].

Research tools, such as PROPHET [5] and FLOODS [6], allow scripting of device equations in a human readable form. Differential operators (e.g., gradients) offer a means of specifying flux terms in the device equations. The source code is available, and new code is required for new operators.

In this paper, we discuss a new device simulator, DEVSIM, which takes a model-based approach. All models are specified as symbolic expressions evaluated each time the program starts. Models are composed of other models and parameters. Dependencies between models are automatically maintained. This hierarchical approach provides a way to manage device model complexity.

Differential operators are not part of the language, as flux evaluation is specified in terms of the models. Commands are embedded in a scripting language, providing flexibility in the simulation [7]. Through the scripting interface, users may:

- modify the way existing models are implemented
- develop models which do not fit into standard formalisms [8]
- create models with arbitrary dependencies

II. Overview

A. Models

Device equations are specified in terms of node, edge and element models in each device region.

1) Node Model: A node model is specified in terms of other node models and parameters. It is evaluated at each node (i.e., vertex) in the simulation mesh. Solution variables are a type of node model.

2) Edge Model: An edge model is specified on the edges connecting pairs of nodes. The expressions are based on other edge models, node models, and parameters.

3) Element Model: Element models allow evaluation of fields on mesh elements. This allows calculation of quantities such as the transverse electric field.

The node model specified for an equation is integrated over the volume of each node in the device region. The flux of the edge model is integrated over the surface area of each node. Contact and interface equations are evaluated using a similar formalism.

B. User Specification

Model expressions are provided by the user and use standard rules of algebraic notation. The derivatives of each model with respect to each solution variable are also models. They may be automatically generated, or specified manually.

For example, the edge model for electric field, $\mathscr{E}$, is specified as

(φ`@n0 -` φ`@n1) * EdgeInverseLength`

where φ`@n0` and φ`@n1` are the potentials at the first and second nodes of the edge, and `EdgeInverseLength` is the inverse distance between them. The simulator generates models for $\mathscr{E}$`:`φ`@n0` and $\mathscr{E}$`:`φ`@n1`, the derivatives of $\mathscr{E}$ with respect to φ at each node.

In contrast to automatic differentiation approaches [9], the user is able to modify the derivative expressions. Special mathematical functions are provided to evaluate flux, such as the Bernoulli function for calculation of current densities [10].

Floating point exceptions are detected so that numerical issues in new models may be debugged. Once debugged, the model may become part of a library for future reuse.

C. Efficiency

We expect efficiency to rival simulators using a source code approach. This is since the parse tree of the model expression is generated once, and reused throughout the simulation. Models whose dependencies have not changed between solver iterations are not reevaluated. The symbolic engine simplifies the expressions for the models, also improving computational efficiency.

Since the software is aware of model dependencies, user-model convergence may be improved in comparison to environments where specific dependencies are assumed [4].

III. Example

A 2D MOSFET was simulated in DEVSIM. The Poisson and carrier-continuity equations were specified through the scripting interface. The Shockley Read Hall (SRH) recombination model [11] was added into the simulation using the expression in Fig. 1. The derivatives required for simulation were generated by the software.

The net doping profile is shown in Fig. 2 for a device with a gate width of 0.5 μm. The potential distribution for $V_{gs} = 2.0$ and $V_{ds} = 0.4$ is shown in Fig. 3. The electron distribution is shown in Fig. 4. The SRH recombination from the model expression in Fig. 1 is shown in Fig. 5.

IV. Conclusion

In this paper, a new TCAD device simulation approach is described. Using a symbolic expression engine, the resulting tool is more flexible than source code approaches, and requires less time for new model development. By taking a hierarchical approach, using standard rules of algebraic notation, and automatically tracking dependencies, the complexity of new model development is reduced.

References

[1] R. W. Dutton and A. J. Strojwas, "Perspectives on technology and technology-driven CAD," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 19, no. 12, pp. 1544–1560, Dec. 2000.

[2] Synopsys, Inc., "Sentaurus Device User Guide," 2007.

[3] "Model development using the C-Interpreter." [Online]. Available: http://www.silvaco.com/products/device_simulation/atlas.html

[4] A. Schenk, P. P. Altermatt, and B. Schmithüsen, "Physical model of incomplete ionization for silicon device simulation," in *IEEE SISPAD*, 2006, pp. 51–54.

[5] "PROPHET." [Online]. Available: http://www-tcad.stanford.edu/˜prophet

[6] "FLOOPS/FLOODS home page." [Online]. Available: http://www.tec.ufl.edu/˜flooxs

[7] J. K. Ousterhout, "Scripting: Higher level programming for the 21st century," *IEEE Computer*, vol. 31, pp. 23–30, 1998.

[8] M. E. Levinshtein and T. T. Mnatsakanov, "On the transport equations in popular commercial device simulators," *IEEE Trans. Electron Devices*, vol. 49, no. 4, pp. 702–703, Apr. 2002.

[9] L. B. Rall, *Automatic Differentiation: Techniques and Applications*, 1st ed. NY: Springer, 1981.

[10] D. L. Scharfetter and H. K. Gummel, "Large-signal analysis of a silicon Read diode oscillator," *IEEE Trans. Electron Devices*, vol. ED-16, no. 1, pp. 64–77, Jan. 1969.

[11] R. S. Muller and T. I. Kamins, *Device Electronics for Integrated Circuits*, 2nd ed. NY: Wiley, 1986.

```
-ElectronCharge*(Electrons*Holes-n_i^2)
/ (τp*(Electrons+n1)+τn*(Holes+p1))
```

Fig. 1. Expression used for SRH recombination. In the expression, `Electrons` and `Holes` are the solution variables, and the other terms are parameters.

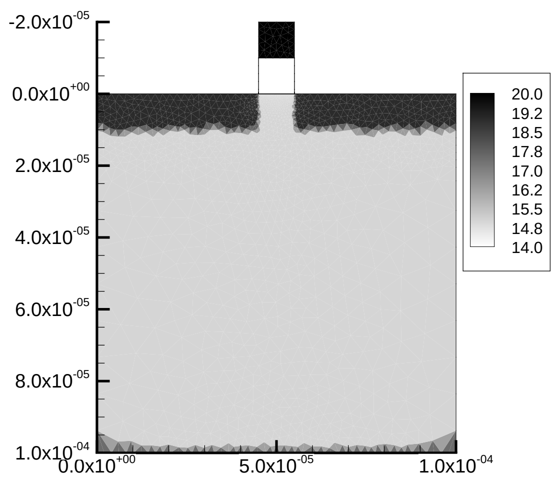

Fig. 2. Net Doping

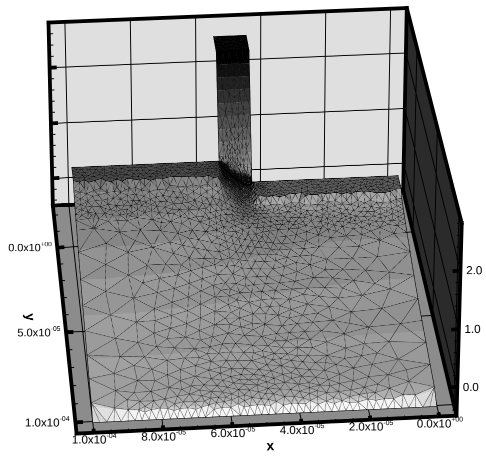

Fig. 3. Potential

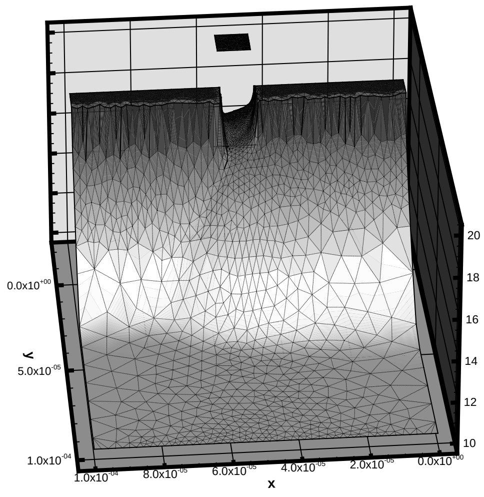

Fig. 4. Electron Density

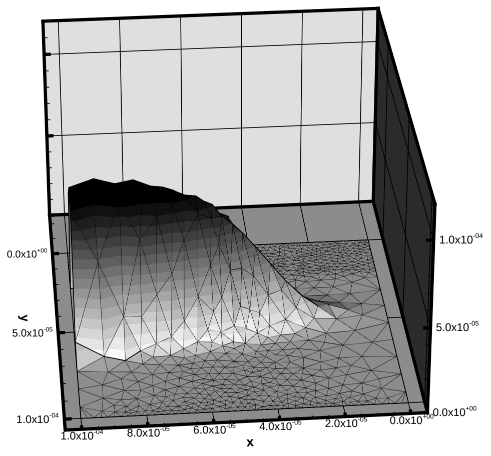

Fig. 5. SRH Recombination