The Journal of Open Source Software

DMT-core: A Python Toolkit for Semiconductor Device Engineers

Mario Krattenmacher[*1, 2], **Markus Müller**[†1, 2], **Pascal Kuthe**[1, 2], **and Michael Schröter**[1, 2]

1 CEDIC, TU Dresden, 01062 Dresden, Germany; **2** SemiMod GmbH, 01159 Dresden, Germany

DOI: DOIunavailable

Software

- Review
- Repository
- Archive

Editor: Pending Editor

Reviewers:

- @Pending Reviewers

Submitted: N/A
Published: N/A

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Statement of need

Semiconductor device engineers are faced by a number of non-trivial tasks that can best be solved efficiently using software. These tasks comprise, amongst others, data analysis, visualization and processing, as well as interfacing (different) circuit and Technology-Computer-Aided-Design (TCAD) simulators. In practice, typically different more or less documented but ultimately similar scripts are employed to solve these tasks. It is not uncommon that fundamental concepts of software engineering, such as Test-Driven-Development (Shull et al., 2010) or the use of state-of-the-art version control tools and practices (Git, CI), are not adhered to by these scripts. This causes severe inefficiencies w.r.t. to cost and time.

The issues inflicted by this practice can be summarized as follows:

- The analysis/visualization/generation of data becomes difficult to re-produce.
- Device engineers work far from their maximum work-efficiency, as they are hindered, instead of empowered, by the employed software infrastructure.
- Knowledge build-up possibly over decades may fade away when developers leave a company or institution.

The Device Modeling Toolkit (`DMT`) presented here aims to solve these issues. `DMT` provides a Python library that offers

- classes and methods relevant for day-to-day device engineering tasks,
- several abstract base classes useful for implementing new interfaces for various types of simulators and
- concrete implementations of the abstract base classes for open-source simulators such as Ngspice (Vogt, 2022), Xyce (Keiter et al., 2014) or Hdev (Müller et al., 2022).

Basic principles of software engineering, such as unit testing, version control and the maintenance of a documentation are adhered to, so that others can also use and contribute to the software.

Summary

`DMT` is implemented as a toolkit that heavily leverages principles ob object-oriented software design. Its Git repository contains documentation, CI jobs that execute unit and integration tests, and create ready to install wheel files. This enables a large community of engineers (with sufficient Python knowledge) to install, use and contribute to the software.

In `DMT` data is stored using `DataFrame` objects. The `DataFrame` class is a subclass of `pandas.DataFrame` (McKinney, 2010), ideally suited for processing and analyzing large amounts of data. `DMT` extends this class with several data-processing methods that are particularly useful

[*] co-first author
[†] co-first author

for electrical quantities such as currents, voltages and charges. Some of these methods are based on routines in `scikit-rf` (Arsenovic et al., 2022).

Electrical data comes from diverse sources like measurements or circuit simulations. A central problem with such data is the naming of variables, which should be consistent throughout the code in order to process data in a unified way. For example, some people might abbreviate the collector current of a bipolar transistor as **I_C**, while others might write **IC** instead. This may lead to major confusion when exchanging data and code with others. `DMT` implements a bullet-proof grammar for naming electrical quantities for solving this problem. During data import all data columns are translated to this grammar. This solves a big issue when transferring data between engineers or even for a single engineer between different work stations and (proprietary) software.

`DMT` offers classes and methods which can be used either directly or need to be subclassed, i.e. for creating interfaces to circuit simulators.

The base class offered by `DMT` for representing electrical devices is called `DutView` (Device-Under-Test). This abstract class provides common attributes and methods that represent measurements, circuit simulations or TCAD simulations. There are several subclasses that add logic:

- `DutMeas` adds logic for DUT instances that contain measured data.
- `DutCircuit` is an abstract class that adds logic for DUT instances that represent circuit simulations. The interface is implemented in the DMT-core module for
  - Xyce (Keiter et al., 2014) in `DutXyce` and
  - Ngspice (Vogt, 2022) in `DutNgspice`.
- `DutTCAD` ads logic for DUT instances that represents devices based on TCAD simulations. The interface is implemented for
  - Hdev (Müller et al., 2022) in `DutHdev`.

Interfaces to other simulators, i.e. proprietary ones, are straight forward to implement. All simulators can be used as drop-in replacements for each other. There are only two necessary steps that need to be implemented for each simulator. First, a routine for generating the simulator input file must be implemented. Second, an import routine that returns a `DataFrame` from the simulator output must be provided. This is illustrated in Figure 1.

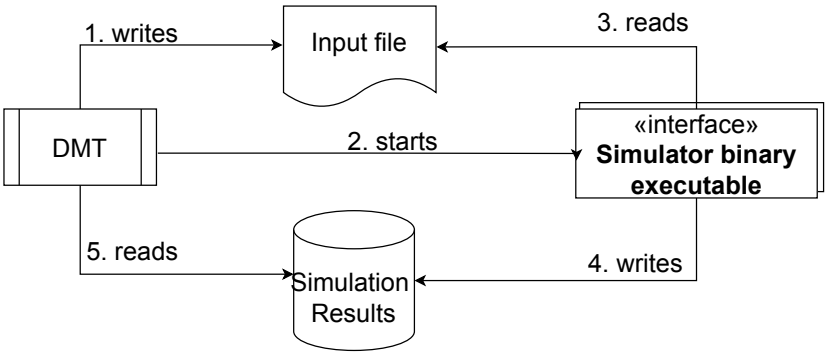

Figure 1: DMT interfacing a circuit simulator and corresponding data flow.

Often one needs to handle many different devices, e.g. transistors with different geometries. For this purpose the `DutLib` class offers a "container" for `DutView` objects, e.g. for storing measurement data of one wafer. A typical use case is loading measurement data generated for a given technology, including specific test structures and transistors.

Circuit and TCAD simulations are started and controlled by the `SimCon` class. This class enables to run many simulations in parallel and utilizes the high core count of modern computers (see Figure 2). Each simulation requires one `DutView` object that defines either a

circuit or TCAD simulation, as well as the definition of a sweep for changing the operating point. The definition of sweeps, i.e. the sweep of voltages or currents, is controlled by objects of the `Sweep` class. `SimCon` generates a hash for every simulation so that simulations need not be run when the software is called multiple times, provided the simulation definition (and therefore the hash) have not changed.

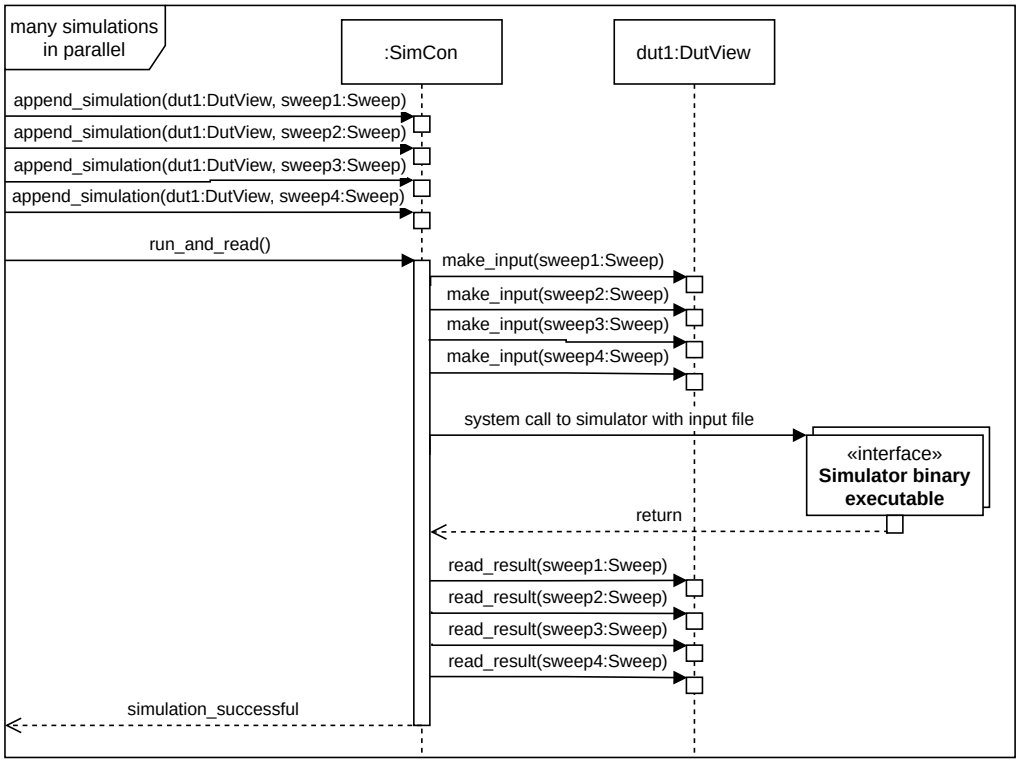

Figure 2: Using DMT to run many simulations in parallel.

Another important class is `MCard`, useful for storing the model parameters of compact models that are defined within Verilog-A files. It implements a container that may store all model parameters, including information on parameter boundaries that is directly obtained from Verilog-A source files. This feature leverages the VerilogAE tool (Kuthe et al., 2020). `MCard` can interpret Verilog-A model codes, save and load lists of model parameters and can also be used to define elements in the `Circuit` class used for defining circuit simulations.

Finally, `DMT` implements the `Plot` class for displaying electrical data using different back-ends:

- `matplotlib` for interactive plots
- `pyqtgraph` for plots to be used in GUI applications
- `LaTeX:pgfplots` for TeX based technical documentation or scientific publications

An example plot of a simulated transistor is shown in Figure 3.

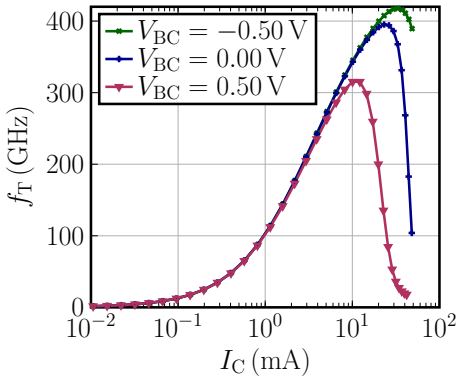

Figure 3: Transit frequency f_T of a Bipolar transistor.

Related Publications

`DMT` is used internally by CEDIC staff in research and by SemiMod for commercial purposes. It has also been used by cooperating institutions and companies. The project has been used in the following publications:

- (Markus Muller et al., 2021): TCAD simulations and plotting.
- (Phillips et al., 2022): Model parameter extraction and TCAD simulation.
- (Weimer et al., 2022): Circuit simulations.
- (M. Muller et al., 2022): Circuit and TCAD simulations.
- (Müller & Schröter, 2019): Model parameter extraction.

`DMT` has been mentioned in Müller et al. (2021).

Related Projects

`DMT` directly uses the VerilogAE tool (Kuthe et al., 2020) for accessing all information in Verilog-AMS files. The TCAD simulator Hdev (Müller et al., 2022) uses the class `DutHdev` as its Python interface.

Acknowledgements

This project would not have been possible without our colleagues Dipl.-Ing. Christoph Weimer and Dr.-Ing. Yves Zimmermann. We particularly acknowledge Wladek Grabinski for his efforts to promote the use of open source software in the semiconductor community.

References

Arsenovic, A., Hillairet, J., Anderson, J., Forsten, H., Ries, V., Eller, M., Sauber, N., Weikle, R., Barnhart, W., & Forstmayr, F. (2022). Scikit-rf: An Open Source Python Package for Microwave Network Creation, Analysis, and Calibration [Speaker's Corner]. *IEEE Microw. Mag.*, *23*(1), 98–105. https://doi.org/10.1109/MMM.2021.3117139

Grabinski, W. (2019). *FOSS TCAD/EDA tools for compact modeling*. Arbeitskreis Bipolar. https://www.iee.et.tu-dresden.de/iee/eb/forsch/AK-Bipo/2019/7-MOS-AK-Association_wgr_BipAK19.pdf

Keiter, E. R., Mei, T., Russo, T. V., Schiek, R. L., Sholander, P. E., Thornquist, H. K., Verley, J. C., & Baur, D. G. (2014). *Xyce Parallel Electronic Simulator Reference Guide , Version 6*

. *2* (September). Sandia National Laboratories (SNL). https://doi.org/10.2172/1826862

Kuthe, P., Müller, M., & Schröter, M. (2020). VerilogAE: An open source Verilog-A compiler for compact model parameter extraction. *IEEE J. Electron Devices Soc.*, *8*, 1416–1423. https://doi.org/10.1109/JEDS.2020.3023165

McKinney, W. (2010). Data Structures for Statistical Computing in Python. *Proc. 9th Python Sci. Conf.*, 56–61. https://doi.org/10.25080/majora-92bf1922-00a

Muller, Markus, Dollfus, P., & Schroter, M. (2021). 1-D drift-diffusion simulation of two-valley semiconductors and devices. *IEEE Trans. Electron Devices*, *68*(3), 1221–1227. https://doi.org/10.1109/TED.2021.3051552

Muller, M., Schroter, M., Jungemann, C., & Weimer, C. (2022). Augmented Drift-Diffusion Transport for the Simulation of Advanced SiGe HBTs. *2019 IEEE BiCMOS Compd. Semicond. Integr. Circuits Technol. Symp.*, 1–4. https://doi.org/10.1109/bcicts50416.2021.9682487

Müller, M., Krattenmacher, M., & Schröter, M. (2019). *Open license parameter extraction tool - Overview and demo for SiGe HBTs*. HICUM Workshop. https://www.iee.et.tu-dresden.de/iee/eb/forsch/Models/workshop_2019/contr_2019/dmt.pdf

Müller, M., Kuthe, P., Krattenmacher, M., & Schröter, M. (2021). *Overview of selected open source tools for compact modeling*. MOS-AK. https://www.mos-ak.org/silicon_valley_2021/presentations/Mueller_MOS-AK_SV_21.pdf

Müller, M., Mothes, S., Claus, M., & Schröter, M. (2022). Hdev: A 1D and 2D Hydrodynamic/Drift-Diffusion solver for SiGe and III-V HBTs. *J. Open Source Softw.*

Müller, M., & Schröter, M. (2019). *Selected Results of HICUM Paramter Extraction for InP HBTs*. Arbeitskreis Bipolar. https://www.iee.et.tu-dresden.de/iee/eb/forsch/AK-Bipo/2019/10-CEDIC_mmu_BipAK19.pdf

Phillips, S., Preisler, E., Zheng, J., Chaudhry, S., Racanelli, M., Muller, M., Schroter, M., McArthur, W., & Howard, D. (2022). Advances in foundry SiGe HBT BiCMOS processes through modeling and device scaling for ultra-high speed applications. *2021 IEEE BiCMOS Compd. Semicond. Integr. Circuits Technol. Symp.*, 1–5. https://doi.org/10.1109/bcicts50416.2021.9682485

Shull, F., Melnik, G., Turhan, B., Layman, L., Diep, M., & Erdogmus, H. (2010). What do we know about test-driven development? *IEEE Softw.*, *27*(6), 16–19. https://doi.org/10.1109/MS.2010.152

Vogt, H. (2022). *Ngspice, the open source Spice circuit simulator - Intro*. http://ngspice.sourceforge.net/

Weimer, C., Sakalas, P., Muller, M., Fischer, G. G., & Schroter, M. (2022). An Experimental Load-Pull Based Large-Signal RF Reliability Study of SiGe HBTs. *2021 IEEE BiCMOS Compd. Semicond. Integr. Circuits Technol. Symp.*, 1–4. https://doi.org/10.1109/bcicts50416.2021.9682473