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: 10.21105/joss.04298

Software

- Review
- Repository
- Archive

Editor: Lucy Whalley

Reviewers:

- @dilawar
- @phoebe-p

Submitted: 08 March 2022
Published: 04 July 2022

Statement of need

Semiconductor device engineers are faced by a number of non-trivial tasks that can be solved efficiently using software. These tasks include, amongst others, data analysis, visualization and processing, as well as interfacing various circuit and Technology-Computer-Aided-Design (TCAD) simulators. In practice, custom 'home-made' scripts of varying quality are employed to solve these tasks. It is often found that fundamental software engineering concepts, such as Test-Driven-Development (Shull et al., 2010), or the use of state-of-the-art version control tools (e.g. Git) and practices (e.g. continuous integration, CI), are not utilized by these scripts.

The issues inflicted by this practice include:

- The analysis/visualization/generation of data becomes difficult to reproduce.
- Device engineers work far from their maximum work-efficiency, as they are hindered, instead of empowered, by the software infrastructure.
- Knowledge built-up, possibly over decades, may be lost 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 to commonly used device engineering tasks
- several abstract base classes for implementing new interfaces to various types of simulators
- 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).

`DMT`-based simulations allow data generation, workflow implementation and visualization to be implemented in a single file, enabling more efficient cooperation and more reproducible research (Stodden et al., 2016). Basic principles in software engineering, such as unit testing, version control, and documentation, are adhered to so that others can use and contribute to the software.

Summary

`DMT` is implemented as a toolkit that follows the principles of object-oriented software design. The `DMT` Git repository contains the `DMT` code alongside documentation and a number of CI routines. These routines execute unit tests, execute integration tests and create ready to install wheel files. This enables electrical engineers with some basic experience in Python to install, use and contribute to the software.

[*]co-first author
[†]co-first author

`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 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 can be generated using a diverse array of methods, ranging from experimental measurements to circuit simulations. A central problem with this data is the inconsistent 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 can lead to confusion when transferring data between engineers, or even for a single engineer transferring data between different work stations and/or (proprietary) software packages. To solve this problem `DMT` implements a bullet-proof grammar for naming electrical quantities, and translates all data columns to this grammar during data import.

`DMT` offers classes and methods which can be either used directly or subclassed, for e.g. 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, e.g. 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.

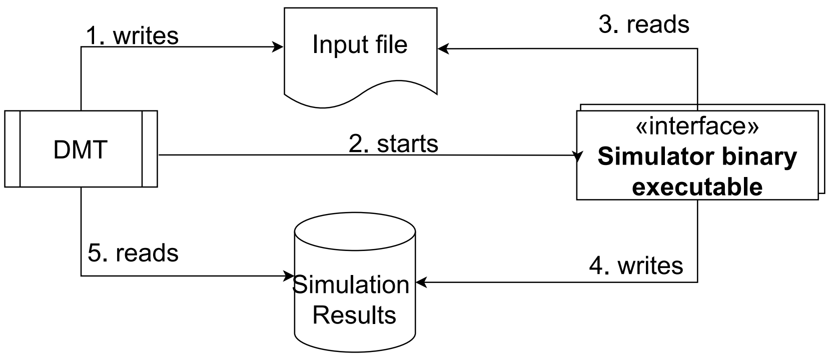

Figure 1: DMT interfacing a circuit simulator and corresponding data flow.

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 for e.g. storing the 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 the user to run many simulations in parallel and utilizes the high core count of modern computers. 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, e.g. the sweep of voltages or currents, is controlled by objects in the Sweep class. SimCon also generates a hash for every simulation, so that identical simulations will not re-run when the software is called multiple times, provided the simulation definition (and therefore the hash) have not changed.

Model parameters are stored using the MCard class. Mcard implements a container for storing 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 2.

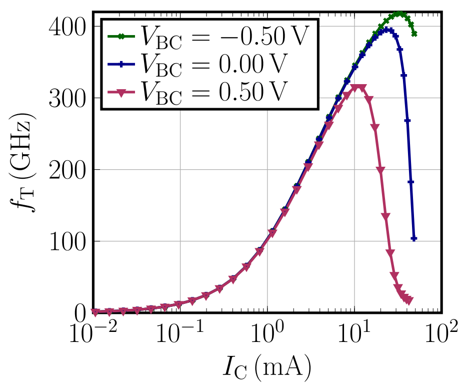

Figure 2: Transit frequency f_T of a Bipolar transistor.

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 contexts:

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

In addition, DMT has been cited in (Grabinski, 2019; Kuthe et al., 2020; Müller et al., 2019, 2021).

Related Projects

DMT directly uses the VerilogAE (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

Stodden, V., McNutt, M., Bailey, D. H., Deelman, E., Gil, Y., Hanson, B., Heroux, M. A., Ioannidis, J. P. A., & Taufer, M. (2016). Enhancing reproducibility for computational methods. *Science*, *354*(6317), 1240–1241. https://doi.org/10.1126/science.aah6168

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