Open Automation Framework for Complex Parametric Electrical Simulations

Sergio Vinagrero Gutierrez, Pietro Inglese, Giorgio Di Natale, Ioana Vatajelu

Open Automation Framework for Complex Parametric Electrical Simulations

Sergio Vinagrero Gutiérrez, Pietro Inglese, Giorgio Di Natale, Elena-Ioana Vatajelu

Univ. Grenoble Alpes, CNRS, Grenoble INP*, TIMA, 38000 Grenoble, France
{sergio.vinagrero-gutierrez,pietro.inglese,giorgio.di-natale,ioana.vatajelu}@univ-grenoble-alpes.fr

***Abstract*—The need to achieve statistically relevant results in electrical simulations requires a large number of iterations under different operating conditions. Moreover, the nature of parametric simulations makes the collection and filtering of the results non-trivial. To tackle these issues, scripts are normally used to control all the parameters. Still, this approach is usually ad-hoc and platform dependent, making the whole procedure hardly reusable, scalable and versatile. We propose a generic, open-source framework to generate complex stimuli and parameters for electrical simulations, together with a programmable Spice- and Verilog-A-based module capable of observing and logging internal states of the circuit to facilitate further result analysis.**

***Index Terms*—electrical simulation, parametric, Verilog-A, Spice**

I. Introduction

Design complexity, ultra-low-power requirements, reliability, robustness and security are becoming increasingly important concerns when designing electronic systems. Moreover, designers are facing novel challenges related to aggressive CMOS technology scaling and the introduction of beyond-CMOS devices (CNT-FET, Memristive, Spintronic). Indeed, several issues must be considered at design time such as fabrication-induced variability, technology-dependent defects, extreme operating/environmental conditions, stochastic behaviours, aging, and possible perturbations (noise, radiations, malicious attacks).

For consecrated technologies and applications, the electrical-level issues are well-understood and easily translatable to higher levels of abstraction (RTL, micro-architectural, system), thus enabling very fast and accurate simulations. However, with the aforementioned issues faced by today's designs, electrical-level simulations are unavoidable since they allow designers to accurately model and understand the behaviour of the target system, and to extrapolate physical-level properties to a higher level of abstraction.

In order to explore the behaviour of an electrical circuit under different conditions and parameters, multiple simulations are performed with the desired setting. In today's circuits based on new technologies, the number of parameters and their interdependencies can be prohibitively large to be exhaustively studied. Monte Carlo [1] and parametric simulations are the classical approaches to cope with this type of problem. Nevertheless, algorithms implemented in industrial CAD tools do not offer the designer full control and observability of desired parameters. Moreover, the range and the distribution of parameters are defined in technological libraries (provided by silicon foundries). However, these data are not easily available for emerging technologies (or at least not yet embedded in industrial libraries).

We propose a unified framework which works for both statistical and parametric simulations performed independently or concomitantly. This allows the designer to fully control the range of parameters and fully observe the circuit behaviour. Moreover, it allows correlating the circuit behaviour with the parameters used at simulation time. This framework implements the following steps:

1) Define the set of parameters relevant to the study. Here we can target process variability, operating/environmental conditions, stochastic behaviours, aging, perturbations;
2) Identify the relevant signals that fully characterise the circuit behaviour;
3) The value of each parameter is generated by a user-defined function;
4) For each combination of parameters, a netlist is generated and simulated. The relevant signals provide the information required about the circuit behaviour.

Another important aspect of electrical simulations is the ability to reproduce results. Indeed, there are situations when it is important to repeat the simulations with the same parameters, for instance when performing design-space exploration, defect analysis, evaluation of countermeasures in secure circuits, etc. Moreover, in the scientific research environment, the repeatability of experiments and results is of paramount importance. However, since several commercial tools exist and their manipulation and operation are not identical, it is difficult to reproduce simulations performed on one tool to another. Most commercial CAD tools provide utilities to generate parameters but in some cases, it is very difficult to replicate the environment. Moreover, there is also the problem of incompatibility between simulators and licenses. Even if the tools and license requirements are met, the aggregation of results after complex parametric simulations is cumbersome and takes a big part of the project time just to prepare the data.

In this paper, we show an open-access and open-source framework to carry out complex parametric electrical simulations and analyses. This framework is a wrapper to commercial tools, which allows researchers to perform and repeat a large

*Institut National Polytechnique Grenoble Alpes

number of simulations, no matter the underlying simulation environment. The framework is implemented in Python and Verilog-A and it is licensed under the MIT License. The code and documentation can be found online [2], [3].

This paper is organised as follows: the current state of the art is summarised in section II, followed by the motivation of this project in section III; In section IV the two parts of the framework are described in detail and some use cases are provided in section V.

II. State of the art

The main format used to describe electrical netlists is called Spice [4], which stands for *Simulation Program with Integrated Circuit Emphasis*. An electrical simulator will take the spice netlist, parse it and simulate it. The same netlist can be simulated under different compatible simulators as it is just a description of the circuit. All of the available tools for electrical simulations provide a graphical user interface to generate circuits based on a drag-and-drop mechanism which is very user-friendly. The graphical user interface can be also used to customise every aspect of the simulation to provide the necessary parameters to the simulator. However, not every available tool provides a mechanism to easily save this configuration to be used later, or on a different computer. Therefore, in order to share a project between different teams, the exact same configuration of the project and sometimes even the operating system have to be precisely copied, which is an unreasonable task to do. Moreover, this problem grows larger with the complexity of the circuit.

CAD tools automatically perform the conversion between the graphical schematic and the textual netlist. However, the other way around is not always performed. Moreover, there are some tools available online that allow parsing netlists or converting results from one file format to another, but these tools are limited since parsing netlists is contextual and simulator-dependent. The same is applied for the file formats containing simulation results as most of the time a special tool is needed to view the results, which makes aggregating results challenging. An example of an automated CAD tool is the AWR Design Environment (AWRDE), provided by Cadence, working in a custom language called SAX. This tool works via a simple interface where the user can control any aspect of the environment through code. Users have also access to powerful tools that allow them to modify part of the circuit and generate custom reports programmatically, and even perform analysis in Python. However, this tool requires an expensive licence that not everyone is able to afford and requires some time to learn to properly use its functionalities.

Other tools like PySpice [5] allow the generation of netlists and the launch of simulations directly with Python. In this way, both stimuli and configuration live under the same block of logic, which reduces the time between iteration cycles. However, this solution is limited to the NGSpice and Xyce simulators and this can have an impact on the workflow.

III. Motivation

The main issues that motivated the creation of this framework are related to the complexity of generating, manipulating and analysing large parametric and statistical simulations and the repeatability of experiments.

There is a massive trend today towards the sharing of information and the enabling of reproducible science. Unfortunately, when it comes to electrical simulations, the use of proprietary CAD tools (which differ from provider to provider) and foundry-licensed device model libraries makes reproducible science almost impossible. This is mainly due to the fact that even having the full description of an electronic circuit, performing parametric or statistical electrical simulations requires CAD tool-specific configurations, which, if not set exactly, might lead to different results than originally reported. This can hinder the repeatability of experiments as scientists are not able to exactly repeat experiments if they are not provided with all the files and settings of the initial research. This framework was created to solve this issue and allow users to create configurations for projects that can easily be shared between different teams and modified to scale up the number of components by exploiting the facility of manipulating text files (Spice netlists). This also makes our framework easily usable with any CAD tool and underlying electrical simulator.

One of the advantages of our proposed framework is that new users don't need to invest time in learning how to use new simulators or get familiar with the user interface since the syntax is minimal and allows very fast results with minor modifications of the files provided by a commercially available tool.

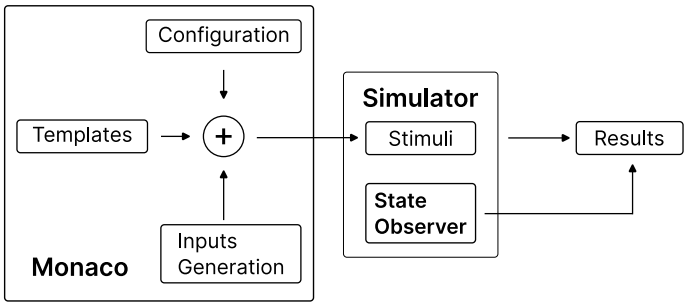

Fig. 1: Functional diagram of the framework. The combination of templates and parameters is done with the framework outside of the simulator, while the state observer has to be placed by the user inside the netlist.

IV. Description of the framework

The framework is divided into two parts. The first part is a spice netlist and simulation configuration builder written in Python. The second one is a Verilog-A module to write the state of an electrical component to a CSV file. Figure 1 shows a small diagram representing how the different parts of the frameworks are combined.

A. Netlist and simulation configuration tool

The main objective of the Python tool is to facilitate the generation of complex electrical parameters as well as inject

them inside predefined spice netlists and simulation configuration files, in a similar manner to a text macro system.

This tool, which from now on will be referred to as *builder*, exploits the way electrical components are defined in spice netlists. For example, a MOSFET transistor is defined as shown below. The syntax can slightly change between different spice implementations, but the definition of the parameters is identical.

```
M1 2 9 3 MOD1 \
L=10u W=5u AD=100p \
AS=100P PD=40u PS=40u
```

Source Code 1: Spice definition of a MOSFET transistor. The slashes indicate that the definition continues in the next line

This builder exploits the textual nature of netlists in order to generate the necessary stimuli for the simulations. A set of values can be generated, inserted into the stimuli templates and feed to the simulator. To achieve this, the user needs to define the different stimuli files that will be used as templates and define the functions to generate the values for each iteration. The different input files are treated as templates and processed by the Jinja [6] template engine. An advantage of using a template engine like Jinja is that logic can be embedded directly into the templates, which means that certain parts of a template can be rendered or discarded depending on the values of some variables.

The builder can be configured through a YAML or TOML config file. This file contains metadata, user-defined properties, the command to execute, the list of files to use as templates and the number of iterations to execute. These variables can be accessed directly in the templates.

In order to generate values, the user needs to define one or multiple `ValueFactories` in Python. These Factories receive the configuration from the YAML file and generate the values for each generation. The user can define as many Factories as desired and they will all be used for every iteration. If one of the Factories however is exhausted (i.e. it doesn't generate more values) the process is stopped. This allows stopping the iterations if some conditions have been met. The user can also define a callback that is run after the command has been executed. This callback has access to the values that have been generated for that iteration and the results of the simulation and can be used for example to perform a clean-up after the simulation.

B. State observer

The state observer is designed for Spice- and Verilog-A-based simulators. It is a sub-circuit that is placed in the schematic and is connected to the components in need of measurements.

For transients simulations, the state observer will write the state measured every time step to a CSV file, so that it can be easily loaded into any conventional software to analyse and extract information. For this reason, the simulation parameter `STEP_SIZE` found in the simulation environment is particularly important: it allows running the state analyser with the desired granularity. Moreover, as shown in the examples section, the state observer can be modified to save the wanted data only in certain cases, to allow a lighter and more application-specific log.

The idea of being able to externally save results in a user-defined file comes also useful in case we want to modify the parameters for the next simulations according to the results just obtained. As an example, we could increase or decrease the step size of parameters depending if we are close or not to the expected working behaviour.

Most of the commercially available software allows exporting simulation results into external files. This process is nevertheless carried out most of the time through a graphical user interface, which slows down the simulation iteration cycle, or the supported file types are not the desired ones.

V. Examples

A. Ring Oscillator-based Physical Unclonable Function

One example where this framework has been used is the reliability analysis of Ring Oscillator-based Physical Unclonable Functions [7]. Due to manufacturing-induced variability, the strength of the inverters in the Ring Oscillator shifts the frequency of oscillation from the nominal value. This effect gets aggravated with voltage and temperature variation. The oscillation frequency depends directly on the number of inverters in the Ring Oscillator. Normally, Ring Oscillators tend to have hundreds of inverters. The vast number of stages, as well as the different environmental conditions, make the process of generating these parametric simulations very convoluted.

In order to obtain a distribution of oscillating frequencies, it is necessary to perform multiple simulations while reproducing process variability.

```
V0 (vdd 0) vsource dc=${vdd} type=dc

{% for s in range(1, props.nstages) -%}
I{{loop.index}} (0 net{{s}} net{{s-1}} vdd) INV \
    vthp={{ params['vthp'][loop.index0] }} \
    vthn={{ params['vthn'][loop.index0] }} \
    lp={{ params['lp'][loop.index0] }} \
    ln={{ params['ln'][loop.index0] }}
{% endfor -%}
```

Source Code 2: Template to generate inverters. The Inverter subcircuit definition has been omitted.

Source code 2 shows a small section of the spice netlist that has been converted to a Jinja template. This part generates a chain of inverters where the number of stages is determined by the user variable `nstages`. In this example, the variability is induced by altering the length and the V_{th} of each PMOS and NMOS. The values are generated using a Factory that draws the values from normal distributions, as seen in the `ParamFactory` shown in the examples in [2]. Thanks to the Jinja templates, the number of inverters for each Ring Oscillator can be embedded in the logic and configured by the user.

Once the schematic is created, the parameters to simulate the process variability are defined in the same way they have been described in section IV-A and are inserted into the netlist.

B. State observer with VTEAM

The state observer has been highly useful in the analysis of memristive-based logic operations for Logic In-Memory-Computing (IMC) [8], [9]. In fact, we noted that IMC with memristors is very sensitive to slight variations. To have a working operation, the IMC operations have constraints on the voltage(s) enabling the operations. Variations of the control voltage(s) and of the duration of the operations itself, as well as memristive values not reaching the nominal value, can affect the behaviour of the operation and not allow it to reach the correct result. For this, it is very important to examine the internal state of memristors at any given time to know if the operations have been properly carried out and understand what affected them in case the result is not optimum.

The VTEAM memristor model [10] provides several parameters to simulate the behaviour of memristors. Being an open Verilog-A description model, it can be easily modified to add new terminals and provide information about interesting internal variables. For this application, it is important to obtain the internal resistance and the distance of the threshold of the doping region. The state observer saves that data, allowing to have a detailed description of the behaviour that can be used to drive in real-time the evolution of the simulation and the proper parametric settings. For all of the memristors that are connected to the state observer, the absolute time of measurement, the internal resistance and the distance of the doping region are written to a parameterised CSV file.

Source code [3] shows an example of the state observer used to save the state of two memristors under the execution of Logic-in-Memory (LIM) operations. The goal of the simulation was to sweep over the control voltage(s) and the duration of the LIM operations to find the best working setting [9]. Hence, for this code, we concatenated the automatic generation of parameters with the state observer. We automatically generated the inputs and the parameters to run the simulation and we saved the simulation times at which each LIM operation is concluded. We also saved the input value(s), the expected output value and selected parameters, including the control voltage(s) and the duration of the operation. During the operation, the resulting file is read by the state observer and it is used to choose the simulation times at which it will save the wanted states. Finally, the resulting CSV generated by the state observer will have all the fields coming from the builder and the read values:

- Provided by the builder
  - Input value(s),
  - Expected output value,
  - Duration of the operation,
  - Control voltage,
  - Timestamp for save action execution.
- Added by the state observer
  - Read value(s).

This allows the user to have all the information about the executed operation in the same CSV and it supports directly performing data analysis from the collected data to assess the correct behaviour of the studied operations.

Another example where this framework could have facilitated the parametric simulations and the data analysis is in the work described in [11], which presents an in-depth analysis of the process, voltage and temperature variability for CIM.

Since the state observer is defined in a Verilog-A file, it can also be integrated with the builder in order to automatically generate the correct number of signals depending on the user-defined configuration. This simplifies the exploration of all the possible space, as the schematic itself will be rendered according to the configuration and the values generated for each iteration.

VI. Conclusions

The framework shown in this article is able to generate complex parametric electrical simulations in a reliable and repeatable manner. This framework is open-access and open-source and works with any available commercial simulator as it doesn't require major modifications to the files used.

This framework could be further expanded by adding an automated system that can generate the state observer for a given amount of different components based on the configuration of the builder.

References

[1] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

[2] S. Vinagrero Gutiérrez and P. Inglese, "Monaco," 2022. [Online]. Available: https://servinagrero.github.io/monaco

[3] P. Inglese, "Verilog-a state observer," 2022. [Online]. Available: https://gist.github.com/servinagrero/b89a173a8a33766d5776035d06207997

[4] L. W. Nagel and D. Pederson, "SPICE (simulation program with integrated circuit emphasis)," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M382, Apr. 1973. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html

[5] F. Salvaire, "Pyspice." [Online]. Available: https://pyspice.fabrice-salvaire.fr

[6] "Jinja." [Online]. Available: https://jinja.palletsprojects.com/en/3.1.x/

[7] S. Vinagrero Gutierrez, G. Di Natale, and E.-I. Vatajelu, "On-line reliability estimation of ring oscillator puf," in *IEEE Electronic Test Symposium*, 2022.

[8] P. Inglese, E. I. Vatajelu, and G. Di Natale, "Memristive Logic-in-Memory Implementations: A Comparison," in *SMACD / PRIME 2021; International Conference on SMACD and 16th Conference on PRIME*, Jul. 2021, pp. 1–4.

[9] ——, "On the Limitations of Concatenating Boolean Operations in Memristive-Based Logic-In-Memory Solutions," in *2021 16th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, Jun. 2021, pp. 1–5.

[10] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "VTEAM: A General Model for Voltage-Controlled Memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, Aug. 2015, conference Name: IEEE Transactions on Circuits and Systems II: Express Briefs P1433: Q15760795 227 citations (Crossref) [2021-02-17].

[11] M. Fieback, C. Münch, A. Gebregiorgis, G. C. Cardoso Medeiros, M. Taouil, S. Hamdioui, and M. Tahoori, "PVT Analysis for RRAM and STT-MRAM-based Logic Computation-in-Memory," Barcelona, May 2022.