Free circuit simulation - SPICE and beyond

Árpád Bűrmen †

† University of Ljubljana, Faculty of Electrical Engineering, Tržaška cesta 25, SI-1000 Ljubljana

E-mail : arpadb@fides.fe.uni-lj.si

ABSTRACT

SPICE is a de-facto standard for free circuit simulation and the role model for most commercial simulators. In this paper we present the SPICE OPUS circuit simulator which is based on the SPICE3 source code. The shortcomings of the original SPICE3 and of its device models are highlighted. We present the solutions to these shortcomings that were implemented in SPICE OPUS. Three promising open-source projects that do not resort to SPICE3 source code are discussed: Gnucap, QUCS, and fREEDA.

Keywords: *SPICE, EDA, CAD, circuit simulation*

1. Berkeley SPICE

The last version of the Simulation Program with Integrated Circuit Emphasis (SPICE) released by the University of California, Berkeley (UCB) was version 3f4 after which the development of the simulator ceased. The publicly available source code was picked up by various organizations and enthusiasts which produced many different SPICE3 [1] based simulators. SPICE OPUS [2] is one of them. The process of creating it took us through the inner workings of SPICE and revealed many bugs and shortcomings of the original SPICE 3f4.

The SPICE3 source code was written in C and meant a great leap forward from SPICE2 (written in FORTRAN) in terms of maintainability, most notably in the area of adding new device models. Unfortunately the code was written by many developers which often failed to follow a common coding practice and often bypassed the defined application programming interface (API). The worst part of the code is the built-in scripting language NUTMEG which has many bugs, quick fixes, and incomplete features. On the other hand the most clean part is the sparse matrix library which is also used in many other simulators.

2. How do they do it?

2.1 Writing down the equations, DC analysis

Simulating a circuit starts with writing down the equations based on two Kichoff's laws and the equations linking branch currents and voltages (branch equations). The Kichoff's current law (KCL) is applied to all nodes with the exception of the reference node (ground). Most simulators today use node voltages as independent variables in the equations. This simplifies the Kirchoff's voltage law (KVL) to simple relations linking every branch voltage to a pair of node voltages.

KVL and KCL equations are linear and include no derivative terms. The nonlinearity and the circuit's dynamics are introduced by the branch equations. In the most commonly used approach to writing down circuit equations the branch voltages from KVL are substituted into branch equations. All branch currents that can be expressed explicitly from the resulting branch equations are substituted into KCL equations. This leaves us with one KCL equation per node (with the exception of the ground node) and some additional branch equations from which the current cannot be expressed explicitly (e.g. voltage sources). The variables in this nonlinear system are mostly node voltages and a few additional branch currents. Assuming that the circuit's elements are not time-dependent the following system of equations is obtained.

$$f(x) + \dot{q}(x) + j(t) \quad = \quad 0 \tag{1}$$

where $\dot{q} = \mathrm{d}q/\mathrm{d}t$, x is the vector of variables, and f, q, and j are nonlinear vector-valued functions. DC analysis assumes that all transients have died out so the $\dot{q}(x)$ term is zero and $j(t)$ is replaced with a constant vector j_0. This leaves us with a nonlinear system of algebraic equations that can be solved using the Newton–Raphson method.

2.2 Small-signal frequency-domain analyses

In the frequency domain (1) is first linearized in the neighborhood of a given operating point x_0 (i.e.

$x(t) = x_0 + \delta x(t)$).

$$G\delta x + C\dot{\delta x} + j(t) \quad = \quad 0 \tag{2}$$

where G and C are the Jacobi matrices of $f(x)$ and $q(x)$ at $x = x_0$. After applying the Fourier transform to (2) we get

$$(G + j\omega C)\Delta X + J \quad = \quad 0 \tag{3}$$

where ΔX and J are the Fourier transforms of δx and $j(t)$. Applying the Laplace transform instead of the Fourier transform to (2) results in an equation similar to (3) which is the basis for the pole-zero analysis. Equation (3) is also used in small signal noise analysis where noise source spectral densities are inserted into J.

2.3 Time-domain analysis

While solving (3) is a matter of a simple LU decomposition and back substitution, the solution of (1) requires the numerical approximation of the derivative terms which is also referred to as numerical integration. The solution to (1) is represented by a sequence of approximate solutions x_k which correspond to discrete timepoints t_k. The derivative term $\dot{q}(t_{k+1})$ can be approximated as $\dot{q}_{k+1}$ by using the following equation.

$$\dot{q}_{k+1} = \frac{A_{-1}}{h_k} q_{k+1} + \sum_{i=0}^{N} \frac{A_i}{h_k} q_{k-i} + \sum_{i=0}^{M} B_i \dot{q}_{k-i} \tag{4}$$

where $t_{k+1} = t_k + h_k$. h_k is also referred to as the timestep. M, N, and the coefficients A_i and B_i are defined by the used integration algorithm (e.g. for the n-th order Gear algorithm $N = n$ and $M = 0$). Note that the coefficients depend on the past timesteps. They must be recalculated at every timepoint if the timestep is not constant.

If (4) is substituted into (1) a nonlinear system of algebraic equations is obtained. Solving this system of equations yields the circuit's approximate solution at the next timepoint x_{k+1}.

Approximation of $\dot{q}(t_{k+1})$ introduces an error (ϵ) into q_{k+1}. This error is also referred to as the local truncation error (LTE) and is proportional to the timestep (h_k).

$$\epsilon = C_{n+1} \frac{\mathrm{d}^{n+1} q_k}{\mathrm{d}t^{n+1}} h_k^{n+1} \tag{5}$$

The error coefficient C_{n+1} can be expressed with A_i, B_i, and the timesteps $h_k, h_{k-1}, ...$

In predictor-corrector algorithms first the circuit's response at t_{k+1} is predicted (e.g. by polynomial extrapolation or explicit integration) and the predicted response is used as the initial point for the Newton-Raphson algorithm. This reduces the number of Newton-Raphson iterations as well as enables us to express the LTE in a more simple manner (without resorting to the $n+1$-th derivative of q at t_k). If the order of the predictor matches the order of the integration algorithm LTE can be expressed as

$$\epsilon = \frac{C_{n+1}}{C_{n+1} - C_{n+1}^P} \cdot (q_{k+1} - q_{k+1}^P) \tag{6}$$

where C_{n+1}^P is the error coefficient of the predictor and q_{k+1}^P is the predicted value of $q(t_{k+1})$.

If the Newton-Raphson algorithm fails to solve the nonlinear system of equations in the prescribed number of iterations the timestep is reduced and the order of the integration algorithm is set to 1. In case of success the new timestep is calculated in such manner that LTE is kept below a predefined bound. If the new timestep sufficiently exceeds the previous one the order of the integration algorithm can be increased.

3. The creation of SPICE OPUS

3.1 Tightly coupled optimization

The original development of SPICE OPUS (up to 2000) was targeted at implementing optimization algorithms for circuit optimization. In those days the computers were much slower than today and the overhead of starting an application and reading a possibly long output file was significant (in 1997 on a Pentium computer the overhead was around 500ms, while on a modern computer it is merely 10ms).

Therefore the idea arose to tightly couple the optimization algorithm with the simulator. This meant extending the simulator with optimization algorithms so that it does not have to be restarted with a new input file every time the optimizer changes some circuit parameter.

3.2 Memory leaks in SPICE3 code

The optimizer was implemented using the built-in SPICE scripting language NUTMEG. The use of NUTMEG quickly revealed its major downside: memory leaks caused by bad programming at UCB. The original SPICE3 was intended for interactive use, but the developers at UCB did not anticipate that an interactive simulator session would include more than hundred simulations. The following simple program which repeats an empty loop 100 times

was eating up memory proportional to the number of repetitions.

```
let const.n=0
while n lt 100
  let const.n=const.n+1
end
```

Memory was allocated without freeing it later because it was easier for the programmers to skip any cleanup actions. While this practice is harmless in batch-mode simulation, it gradually eats up the all available memory when several thousand simulations are performed without restarting the simulator.

After cleaning up the simulator (removing the memory leaks, fixing bugs, and completing incomplete features) we ended up with a robust and reliable simulator which was actually a by-product of our developments in the area of circuit optimization. The simulator quickly gained popularity due to its stability and the fact that there was a Linux version available.

3.3 XSPICE extensions

In 2000 we added the XSPICE extensions to SPICE OPUS. The extensions simplify the process of adding new device models to the simulator, albeit NOISE and PZ analysis are not supported. It is also not possible to create XSPICE devices with internal nodes (or variables for that matter). Another advancement is the introduction of mixed-mode simulation.

In contrast to SPICE3 the XSPICE source code is better organized. Despite the fact that XSPICE was written by several developers the source code adheres to common coding rules. XSPICE extensions are also well documented.

3.4 Simulation of integrated circuits

In 2001 we started our first attempts at integrated circuit (IC) optimization. The first things we had to implement were the `.include` and the `.lib` statement which are found in almost every process model library available. We also had to implement a mechanism for switching between multiple predefined topologies and model libraries (the `.netclass` netlist statement and the `netclass` NUTMEG command).

A much tougher nut to crack was the support for parameterized subcircuits. The original SPICE3 code treats a subcircuit definition as a macro. Subcircuit instances are expanded resulting in a flat circuit which is subsequently parsed by the simulator. Subcircuits are implemented as a preprocessing step so implementing parameterized subcircuits that support interactive modification of parameters (i.e. by using a NUTMEG command) required a rewrite of the whole subcircuit handling code. NUTMEG was used as the language for evaluating the parametric expressions. The obtained values are substituted into instance descriptions upon which descriptions are parsed. This made it possible to keep most of the parser code unchanged.

At the time the BSIM3 MOS model was used in most process model libraries so we added BSIM3 and several other modern MOS models (e.g. BSIM3, BSIM4, BSIM SOI, EKV, ...) to SPICE OPUS. Most of them support the adaptation of MOS parameters (e.g. the threshold voltage) to the dimensions of the transistor channel (i.e. W and L). It is customary that the whole feasible design space in terms of W and L is divided into rectangular areas (bins) delimited by the values of Wmin, Wmax, Lmin, and Lmax. Every bin has a separate model in the process model library. As long as the values of W and L cannot be changed interactively the selection of the correct model (bin) for given W and L can be done by a preprocessor. Because SPICE OPUS is interactive and the user can change the values of W and L for every transistor, the mechanism for migrating a MOS transistor instance from one bin to another had to be implemented.

A common practice in IC design is connecting m identical devices in parallel. For this purpose the M parameter was added to all SPICE devices. It specifies the number of parallel instances. The implementation is fairly simple. In most cases the matrix contributions have to be multiplied by M before they are loaded.

Optimization of ICs requires many fairly complex manipulations of the results for extracting performance measures like bandwidth, phase margin, etc. While other simulators implement these using dedicated description languages (e.g. MDL and OCEAN for SPECTRE, the `.measure` statement in HSPICE, ...) we had the advantage of a built-in language (NUTMEG). With the implementation of facilities for manipulating position markers associated with waveforms (the `cursor` NUTMEG command and the `[%]` operator) we obtained the equivalent of the post-processing facilities available in other simulators.

3.5 Simulator convergence

During the many IC optimization runs with various process model libraries we often ran into convergence problems (i.e. failure to solve the nonlinear system of equations with a limited number of iterations). In such cases SPICE falls back to continuation methods (homotopy methods). In SPICE3 both continuation methods (GMIN stepping and source stepping) were implemented incorrectly which resulted in an excessive number of circuits not converging, even when the number of available iterations (GMIN and source steps) was increased. The problem was solved by replacing the fixed steps with adaptive stepping.

When continuation methods fail SPICE OPUS attempts to solve the system using the damped Newton algorithm. One Newton step for solving $f(x) = 0$ can be written as:

$$x_{n+1} = x_n - (\mathrm{d}f/\mathrm{d}x)^{-1} f(x) \tag{7}$$

The second term is the update term which is reduced by a factor $0 < \alpha < 1$ when the damped algorithm is used. The number of iterations required for the algorithm to converge increases while the algorithm becomes more robust and many problems on which the original algorithm fails can now be solved.

A common way for obtaining the DC operating point of troublesome circuits is to attach capacitors to ground at every node and then gradually ramp up all nonzero voltage and current sources while performing a transient (time-domain) analysis. For stable circuits the signals converge to the DC operating point after a sufficiently long simulation time. In SPICE OPUS this approach is termed "source lifting".

XSPICE also brings its own (naïve) approach to solving convergence problems termed "shunting". Shunting involves resistors that are connected from every node to ground. If the resistances are large enough the resistors improve convergence and the results are close to the true solution. This approach is the equivalent of a single GMIN step.

Since there are several algorithms and simulator parameters available for solving convergence problems a parameter tuning algorithm was implemented in SPICE OPUS which automatically solves most convergence problems.

3.6 Infinity, NaN, and other bugs

The only bug we found in the sparse matrix library was associated with the inversion of denormalized double precision floating point numbers. If one tries to invert 10^{-320} the result is always infinite because the largest possible double precision number is approximately 10^{308}. Any values below 10^{-308} must be treated as zero.

Infinite values are no problem per se, as long as they get annihilated in operations like $e^{-\infty}$ or $1/\infty$. A problem arises when ∞/∞ or $0 \cdot \infty$ occurs. The result is "not a number" (NaN). Once a NaN is obtained it poisons every subsequent calculation which also results in a NaN.

NaN values can also occur due to an optimizing compiler rearranging mathematical expressions to save some computational time. Expressions not written with floating point limits in mind can result in infinite or NaN values when some extreme conditions occur. This was the case with the BSIM3 model in Linux. Because we failed to identify the source of the problem (debugging optimized code is theoretically possible, but extremely hard in practice), the only solution was to compile the BSIM3 model without code optimization.

The code for determining the transient time step had to be revised thoroughly. The calculation of the LTE coefficients as well as the calculation of the LTE was incorrect. Beside this there were several bugs in the transient analysis that resulted in "timestep too small" errors.

Minor bugs were removed in various parts of the simulator, e.g. setting of initial conditions, setting of nodesets, breakpoint handling, etc.

3.7 The audience of SPICE OPUS

The largest part of SPICE OPUS' audience comes from educational institutions (38%). The reason for this is probably the stability and the availability of documentation and examples. 33% of users come from research and development. A large part of users (22%) uses SPICE OPUS for personal purposes. There are also some government/military users (1%). The rest of the downloaders (6%) didn't specify the line of work they are in.

4. Beyond SPICE

There exist several recompilations of the original SPICE3 source code. All of them are plagued by the same problems as SPICE3 is. The SPICE3 source code is fairly open when it comes to adding new device models. Unfortunately the API for defining a model requires the implementation of a separate matrix loading function for every type of analysis. This

means that any new analysis (like harmonic balance) that is added to SPICE3 simulators requires the implementation of the analysis-dependent matrix load function for every existing device.

Several individuals and groups have accepted the challenge of designing and implementing an open-source circuit simulator. In this section we limit ourselves to open-source projects. Such projects have the potential to spread the development beyond a small and usually closed group of people. Even if the original team stops the development an open source project can easily be continued by some unrelated team of people. We take a closer look at three such simulators: GNUcap (ACS), QUCS, and fREEDA.

4.1 Gnucap (formerly ACS)

Gnucap [4] was started with the aim to implement a circuit simulator in C++. Originally the project was named ACS. The first version dates back to 1992. The simulator supports DC, small signal AC, and transient analysis. The API for model implementation is not very user-friendly when compared to QUCS and fREEDA.

Gnucap has a model compiler which makes it possible to describe a device model in a more concise manner. For instance the implementation of the BSIM3 MOS model for the model compiler takes 2800 lines of code, while only the transient matrix load function for SPICE3 is more than 3200 lines long. The model compiler is oriented toward subcircuit level modeling. No support for automatic differentiation is available.

Latest developments enable Gnucap to include standard SPICE3 device models with almost no changes to the source code. This makes it possible for Gnucap to take advantage of the latest BSIM models, since the implementation of these models for the SPICE3 simulator is freely available.

Gnucap has built-in capabilities for event-driven simulation and also supports mixed-mode simulation. Unfortunately there are not many event-driven device models available for Gnucap nor there is any support for VHDL or Verilog.

Gnucap is a project that has been around for a long time. The progress of the project seems to be slow but steady. The device models are implemented in a quite cryptic fashion and some good documentation of simulator's internal workings would be extremely welcome. The architecture of the simulator is not very friendly when it comes to adding new types of analysis. If Gnucap will rely on the source code of SPICE3 BSIM models, it will become even harder to add new types of analysis (e.g. harmonic balance).

4.2 QUCS

The first version of the Quite Universal Circuit Simulator (QUCS) [5] was released in 2003. Although QUCS is still in early stage of development it already has many important features implemented. The simulator supports DC, small signal AC and noise analysis, transient analysis, and hopefully in the future harmonic balance analysis.

Contrary to Gnucap QUCS does not derive its device models from simpler base classes. Both subcircuit-based device modeling as well as describing the devices with their respective matrix stamps (like in SPICE3) are supported.

Every analysis has its own matrix load function which introduces the many drawbacks of SPICE3. Fortunately QUCS supports ADMS which can compile a model described in Verilog-A to a C++ implementation using QUCS' API. ADMS provides automatic differentiation which makes model development much simpler than in Gnucap.

QUCS is still missing advanced MOS models (e.g. BSIM). Theoretically one could use the publicly available SPICE3 BSIM models since the basic architecture and device modeling in QUCS are similar to those found in SPICE3. Of course this would bring the same disadvantages as it does in Gnucap.

Event-driven simulation is possible hrough the use of FreeHDL and Icarus Verilog but at this point QUCS does not support mixed-mode simulation.

There are also some drawbacks resulting from the fact that QUCS is still in early stages of development. The simulator does not use sparse matrices which makes it questionable when medium and large size circuits are simulated. Instead of using established and verified standard mathematical packages for solving linear and nonlinear equation systems, Fourier transformation, etc. QUCS uses its own implementations.

Compared to Gnucap QUCS has a more active development team which is constantly improving the simulator.

4.3 fREEDA

fREEDA [6] is a simulator with a significantly different approach than other simulators described in this paper. The approach used for device modeling is based on state variables. The system of equations

in fREEDA includes the state variables as well as the node voltages. Unfortunately the way devices are modeled in fREEDA complicates the implementation of charge-conserving models [3].

There is no support for ADMS, but on the other hand the simulator supports automatic differentiation through operator overloading. The way a new device model is described is very straightforward and concise (the BSIM3 model implementation takes less than 800 lines of code).

The whole structure of fREEDA is oriented toward modularity, which means that new analyses can be added without the need to rewrite the device models. The simulator supports DC, small-signal AC, transient, and harmonic balance analysis. C++ language features are used for simplifying the implementation of device models and avoiding separate matrix load functions for every type of analysis.

At the present there is no support for event-driven and mixed-mode simulation in fREEDA.

fREEDA uses well established mathematical libraries for handling dense and sparse matrices, solving linear and nonlinear systems, Fourier transformation, etc. New releases of fREEDA are infrequent (the last one is dated January 2009).

5. Conclusion

Despite its age SPICE3 is still the most popular open-sorce simulator that supports the latest device models. If, however, one needs a simulator that supports RF analyses, like the ones found in commercial simulators, there is no real open-source alternative.

SPICE3 and its derivatives have some major drawbacks. First of all the simulator code has many bugs which are more or less successfuly fixed in SPICE3 based simulators. The device modeling approach leaves little space for extending the simulator with new kinds of analysis without rewriting or at least extending the source code of all existing device models. SPICE3 is written in C and therefore cannot take advantage of advanced language features provided by C++.

Recent developments in the area of open-source circuit simulation have brought fresh approaches to the problem of designing a simulator. Most newly developed simulators use C++ as the programming language. C++ language features have great potential for reducing the amount of programming needed for implementing a simulator.

The need for a more simple approach to device model implementation has also been recognized. In QUCS ADMS is used for compiling Verilog-A models into C++ code. On the other hand fREEDA has a thought-out API which greatly simplifies the implementation of device models.

An open-source simulator with a well defined API for adding new models and analyses would bring several benefits to electronic engineering community. The separation of the models from the simulator would make it possible to add new analyses without the need to rewrite all existing device models. This would give researchers a powerful platform for developing new methods of analysis that would become immediately available to anyone interested in using them.

Acknowledgements

The research has been supported by the Ministry of Higher Education, Science and Technology of the Republic of Slovenia within programme P2-0246 - Algorithms and optimization methods in telecommunications.

References

[1] T. L. Quarles. Analysis of Performance and Convergence Issues for Circuit Simulation. Memorandum No. UCB/ERL M89/42. University of California, Berkeley, 1989.

[2] T. Tuma., Á. Bűrmen. Circuit Simulation with SPICE OPUS: Theory and Practice. Birkhäuser, Boston, USA, 2009.

[3] P. Yang, B. D. Epler, P. K. Chatterjee. An Investigation of the Charge Conservation Problem for MOSFET Circuit Simulation. IEEE Journal of Solid-State Circuits, 18: 128–138, 1983.

[4] A. T. Davis. An Overview of Algorithms in Gnucap. Proceedings of the 15th Biennial University/Government/Industry Microelectronics Symposium, 360–361, 2003.

[5] M. E. Brinson, S. Jahn. Qucs: A GPL software package for circuit simulation, compact device modeling and circuit macromodeling from DC to RF and beyond. International Journal of Numerical Modeling (IJNM): Electronic Networks, Devices and Fields, 22, 297–319, 2008.

[6] M. B. Steer, N. M. Kripliani, S. Luniya, F. Hart, J. Lowry, C. E. Christoffersen. fREEDA: an open source circuit simulator. Proceedings of the 2006 Workshop on Integrated Nonlinear Microwave and Millimeter-wave Circuits, 2006.