Memristor Software Simulation

Cameron Allen

Sponsor: Sameer Sonkusale
Advisor: Ron Lasser

Abstract

In 2008, a research team at HP announced that they had discovered a new circuit element – the memristor [2]. This discovery was arguably as important as that of the resistor, capacitor, or inductor. Since the HP announcement, researchers and engineering professionals have been trying to understand the new technology and to develop potential applications. To do this, they need robust circuit simulation software that allows them to explore how memristors work alongside existing components. This project surveyed the available simulation tools and ultimately introduced a new one. The end result was a modified version of SPICE that had built-in support for memristors, and which was in several ways better than the existing software programs.

Introduction

The memristor, short for "memory resistor", is a fourth type of basic circuit element. It is a two-terminal device whose behavior is characterized by a relationship between the time integrals of current and voltage. This type of relationship can be observed in a wide variety of nanoscale devices, but it was first understood in metal-oxide crossbar switches [2]. True to its name, the memristor operates as a sort of resistor with memory. In general, the resistance of the device is constant except in the presence of an electric field: if a positive voltage is applied across a memristor, the resistance increases, and if a negative voltage is applied, the resistance decreases. Because of this unique behavior, there is speculation that memristors could have applications from high-density, non-volatile memory to modeling brain synapses in neural networks [5, 6].

For now, the biggest obstacle to memristor application development and circuit design is that memristors are so new. Traditional circuit theory doesn't apply to memristors in the majority of situations, and although memristors are fairly easy to fabricate, it can be difficult to acquire a large number of them. For now, this means software simulation is probably the easiest way to explore memristor behavior. At the beginning of this project, all of the freely available simulation models had significant limitations, and none of them could reliably model memristor behavior. The goal of this project was to create a simulation tool that worked with existing circuit components, but which also had strong support for modeling memristors, across a wide variety of circuit classes.

Problem Background

Memristors can be fabricated in a number of ways, but perhaps the most widely known version is the platinum/titanium-oxide/platinum memristor that HP developed in 2008 [2]. In their device, the titanium-oxide layer had oxygen vacancies on one side that acted as charge carriers and improved conductance (see Fig. 1a). These oxygen vacancies had an effective positive charge, and unlike traditional semiconductor devices, they were not fixed in place due to lattice bonds. This meant that in an electric field, the oxygen vacancies were free to move throughout the length of the device, even into the pure region of the metal-oxide.

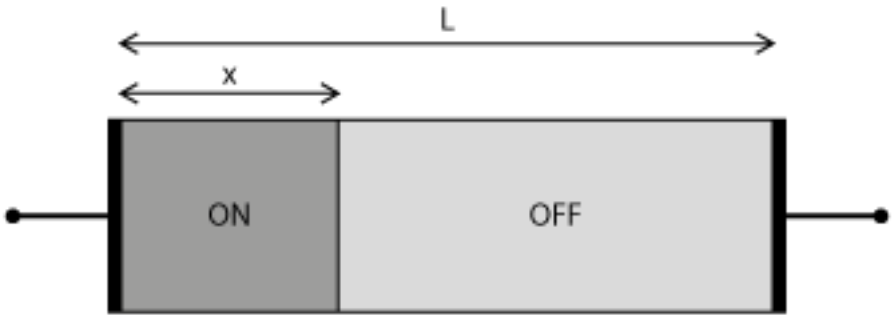
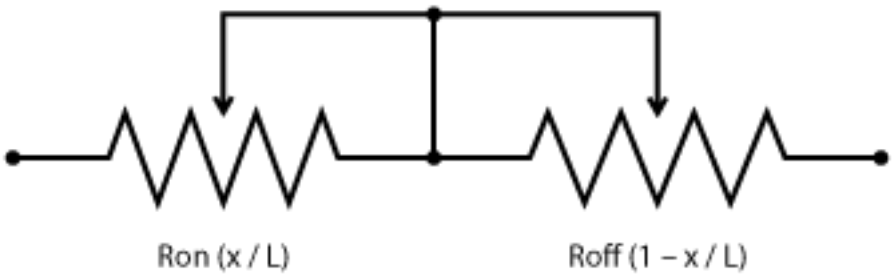


Fig. 1a – Diagram of Memristor Doping Profile

Fig. 1b – Memristor Equivalent Circuit Model



The simplest equivalent model for HP's device is two variable resistors, connected in series (see Fig. 1b): a resistor R_{ON} that is fairly small, representing the oxygen-depleted layer of the oxide; and a resistor R_{OFF} that is several orders of magnitude larger, representing the pure layer [2]. If L is the length of the device, and x is the length of the doped (oxygen-poor) region, then an expression for the total resistance is given by:

$$R = R_{ON}\frac{x(t)}{L} + R_{OFF}\left(1 - \frac{x(t)}{L}\right) \qquad \text{(Eqn. 1)}$$

The resistance then, is a function of the current doping profile of the device. Eqn. 1 assumes that the doping profile is approximately a step function, but the result could certainly be generalized to more complicated profiles.

It is important to notice that the doping profile is time-dependent. The rate that the profile changes is limited by how fast the vacancies can move. When put in terms of electrical mobility:

$$\frac{dx}{dt} = \mu_v E \qquad \text{(Eqn. 2)}$$

where μ_v is the mobility of the vacancies, and E is the electric field. Eqn. 2 does not take into account the boundary conditions: as x approaches 0 or L, dx/dt should approach zero, since the vacancies cannot move past the ends of the device. This means the resistance is always in the range $[R_{ON}, R_{OFF}]$, regardless of the input voltage.

There are a number of papers circulating that describe ways of modeling this type of memristor behavior with software [3, 7, 8]. However, only one of these models was found to be compatible with any of the latest versions of the SPICE circuit simulator [7]. After some testing, the one model that did work was found to be inadequate for modeling memristors in situations where the length of the doped region (x) approached either boundary. The simulation would become unstable and would produce results that did not make physical sense. A simulation that could accurately model these boundary conditions and that still supported existing circuit components would certainly have value for those interested in learning about or developing memristor circuits.

Method of Solution

Exploring the Alternatives

The beginning of the project was mostly comprised of researching memristors and existing simulation methods, including both hardware and software. A significant portion of time was spent exploring simulation programs that were not compatible with the available versions of SPICE. The only sub-circuit model that SPICE could interpret was the one by Rák and Cserey []. Unfortunately, this model was not able to simulate a simple memristor/voltage source circuit at low frequencies or DC, because the component responsible for keeping track of the state variable (the capacitor *Cmem*) could not appropriately handle the boundary conditions. The simulation would fail with an "out of range" error and report that the time step was too small. No time step was found to be small enough to prevent this error, and it would occur any time the memristor state variable reached its physical limits.

To try to remedy this problem, several alternative simulation options (including SPICE, MATLAB, Simulink, and C/C++) were evaluated based on functionality, ease of use, portability, and strength of existing circuit modeling support. SPICE was determined to be the best

alternative – that is, the SPICE program itself, not just another sub-circuit implementation. SPICE was already one of the industry standard analog circuit simulation programs, and because it was also open source, it quickly became the clear choice for implementing a custom memristor module.

Configuring SPICE for Mac OS X

The first step in writing a module for SPICE was to make sure it would compile from source []. This involved modifying configuration files, changing system-specific file paths, and linking against the appropriate libraries. The system chosen for development was one running Mac OS X 10.6, because of convenience and the fact that the operating system was Unix-based. However, since the newest version of Berkeley SPICE (3f5) was written over 10 years ago, the existing "mac" setting was inevitably out-of-date. A new setting called "macosx" was created and configured to allow SPICE to compile on the newer operating system.

Copying an Existing Component

Once SPICE was properly configured, compiled, and verified with some example circuits, the next step was to begin the process of creating a memristor module. The SPICE code was written in C, and it was divided into a number of modules already. Indeed, all of the native components for SPICE had their own separate folders. There were also modules for input parsing, various numerical methods, and the simulator itself. It seemed logical to copy an existing component and modify its behavior to emulate that of a memristor, rather than trying to write one from scratch. The obvious choice of starting component was the resistor, since it had by far the closest behavior to a memristor of all the native SPICE components.

The resistor module "res" was copied and renamed "mem", and all references to resistors inside were replaced with their memristor equivalents. It was also necessary to add the "mem" library to the configuration files, as well as to add new input parsing logic. Most netlist prefixes in SPICE are reserved for existing components (e.g. "r" for resistor), so the prefix choice ("a") for memristor was based solely on availability. The source code was recompiled to incorporate the custom module for memristors (although so far, they had no difference in behavior from resistors), and then checked with a few test circuits. The testing amounted to replacing "r" with "a" in the netlists, and verifying that SPICE still produced the same output.

Modifying the Load Function

The final step – changing the code to accurately reflect memristor behavior – was the most complicated. Luckily, it turned out that a good portion of the work involved in reporting the resistance value was in one particular file ("memload.c", modified from "resload.c"). The resistor code simply loaded a single conductance value into the four positions of a 2x2 matrix. For short time steps, the new device had to look essentially like a resistor, so there just needed to be a custom function to update the conductance value immediately before it was loaded into the matrix. This update/load functionality was also how SPICE handled more complicated devices such as capacitors and diodes.

Modifying the load function required adding several new model variables to handle things like R_{ON}, R_{OFF}, μ_V, x, and L. There were also some changes to be made to the initialization function. The memristor module was implemented using Equations 1 and 2, and then the code was tested for a simple memristor/voltage source circuit.

But the new module didn't work as anticipated. In fact, the incorrect results that it produced revealed some unexpected properties of the simulation algorithm. It turned out that SPICE used a variable step size, which could be increased or decreased as necessary to improve performance – a fairly standard technique in numerical analysis. This particular variable-step algorithm, however, would frequently revisit time points it had previously seen, if the algorithm happened to decide that it needed a smaller step at those time points. The updates to the resistance value in the memristor module did not take these time-jumps into account. This lack of awareness hadn't mattered for a normal resistor, but since the memristor was so time-dependent, a new problem manifested itself, wherein the simulation tried to revert to a previous time point, but the memristor wrongly kept its newer state.

Handling the "Time-Jumps"

Solving the time-jump problem required storing old resistance values in a data structure that would allow lookups based on specific time values. Ideally the data structure wouldn't require much memory, but more importantly it needed to be able to find resistance values quickly, since it would likely do that a great deal during simulation. The implementation was a modified version of a linked-list of resistance/time pairs, which would be reset after every time a

resistance value was looked up. The implementation took advantage of the fact that none of the time-jumps ever went to a time prior to that of the previous time-jump. To put it another way, once the simulation had requested a particular time/resistance value, the entire rest of the list could be deleted, with no actual loss of data. This allowed for much quicker lookups than would have otherwise been possible, since the program did not have to traverse nearly as many nodes in the list before finding the one it needed.

With this modification to the update function, the memristor/voltage source circuit began simulating as expected (see the Results section). At this stage, there were only a few more minor features to add; namely, support for model input parameters in the netlist and support for a non-linear window function such as the one described in Biolek et al. []. With these adjustments, the memristor SPICE module was effectively finished (as soon as support could be added for operating systems besides Mac OS X).

Results and Analysis

The outcome of the project was a modified version of SPICE with native support for memristors. It was a command-line application that used the standard SPICE netlist syntax (plus a new prefix "a" for memristors) and supported graphical plotting using the X Window System (see Fig. 2). The program also allowed for memristor model parameters to be enumerated in the netlist, in order to specify device length, vacancy mobility, and minimum, maximum, and starting resistance. The module theoretically would work with any existing SPICE component (as well as with combinations of components and sub-circuits), provided that the time step was small enough.

It was impossible to test every circuit configuration, so verification concentrated on simply ensuring that the configurations that had been investigated with the Rák and Cserey simulator would simulate correctly. Firstly, the new program was able to correctly simulate the circuit configurations where the Rák and Cserey simulator had performed as expected (see Fig. 2c). Second, perhaps more importantly, the new program could even handle the configurations that had caused the Rák and Cserey simulator to fail. There were no more "out of range" errors, and the results were consistent with the expectations for a memristor experiencing hard switching –

the same hysteresis loops in the current-voltage graph, except with flattened portions where the resistance reached a maximum or minimum value (see Fig. 2d).

Although the final program was able to simulate memristors better than the Rák and Cserey model, it still had a number of drawbacks. First, it was only built to run on Mac OS X. In order to become widely used as a simulation tool, it likely needed to support either Windows or Linux/Unix (or both). Second, it needed more extensive testing, and full support for temperature, noise, and sensitivity analyses (all of which were ignored in the interest of time). Third, the "a" netlist prefix needed to be changed to something more permanent and professional. The "m" prefix was already being used to mean a generic model, but it could have been extended to allow for memristors as well. None of these were requirements for modeling transients in memristor circuits, but most (if not all) of them would likely need to be implemented for the software to be a complete package.

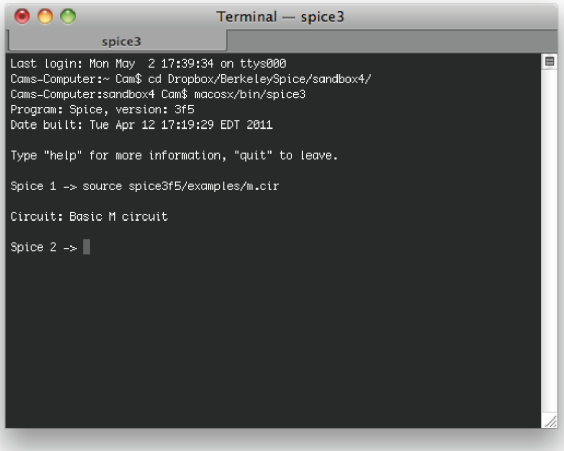


Fig. 2a – Command-Line Interface

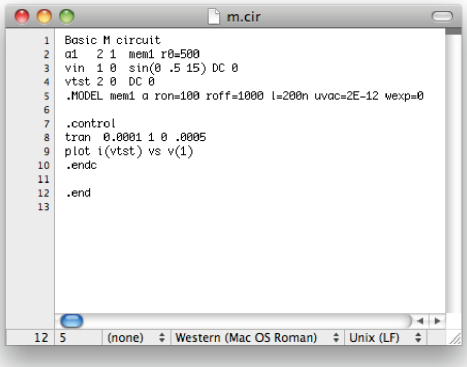


Fig. 2b – Netlist Syntax

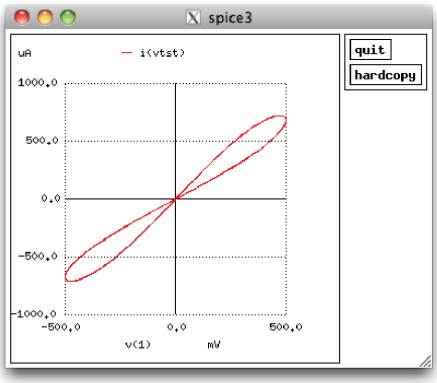


Fig. 2c – Memristor Hysteresis Current-Voltage Loops

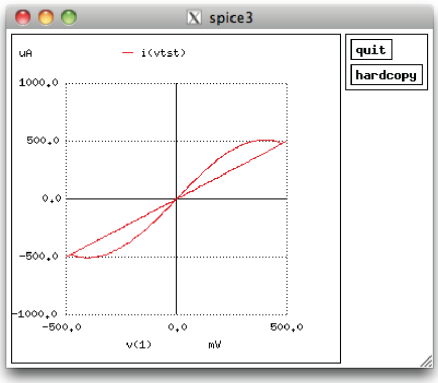


Fig. 2d – Memristor Hysteresis Loops During Hard Switching

Conclusions

The final program showed a great deal of potential as a tool for modeling memristor circuits. It improved on previous software simulation tools by implementing memristors natively, and by accurately modeling memristor circuit configurations that caused previous simulators to fail. There is certainly room for improvement and additional features, but even in its current form, the program can provide significant value to electrical and computer engineers, both in industry and in academia.

Overall the project afforded the opportunity to work with cutting-edge technology at the intersection of analog electronics and software development. It was also a learning experience – one that taught about designing new modules for existing software, as well as debugging and testing that software to make sure it's working correctly. Most importantly, provided the opportunity to practice things like managing large-scale and long-term projects. The end result was a memristor simulation program that in many ways was better than any of the alternatives, and with a few modifications, it may even help to shape the path for memristor circuit development in the years to come.

References

1. "*Memristor – the missing circuit element*," L. Chua. *IEEE Journals* **18**, 507-519 (1971)
2. "*The missing memristor found*," D. Strukov, G. Snider, D. Stewart, R. S. Williams. *Nature* **453**, 80-83 (2008)
3. "*SPICE Model of Memristor with Nonlinear Dopant Drift*," Z. Biolek, D. Biolek, V. Biolkova. Czech Republic University of Defense. (2009)
4. "*Memristive switching mechanism for metal/oxide/metal nanodevices*," J. J. Yang, M. Picket, X. Li, D. Ohlberg, D. Stewart, R. S. Williams. *Nature Nanotechnology* **4**, 429 - 433 (2008)
5. "*How we found the missing memristor*," R. S. Williams. *IEEE Spectrum* (Dec. 2008)
6. "*H.P. Sees a Revolution in Memory Chip-Making*," J. Markoff. *New York Times* (Apr. 2010)

7. "*Macromodeling of the Memristor in SPICE*," Ádám Rák and György Cserey. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **29**, 632 - 636 (Apr. 2010)
8. "*The fourth element: characteristics, modelling* [sic] *and electromagnetic theory of the memristor*," O. Kavehei, A. Iqbal, Y. S. Kim, K. Eshraghian, S. F. Al-Sarawi, D. Abbott. *Proceedings of The Royal Society A* **466**, 2175-2202 (Mar. 2010)
9. "*HP nano device implements memristor*," S. Bush. *Electronics Weekly* (May 2008)
10. "*Memristors…Made of Blood?"* R. Courtland. *IEEE Spectrum* (Apr. 2011)
11. "*Memristors*," C. Allen. Dept. of Electrical and Computer Engineering, Tufts University. (May 2010)