Non-Linear Circuit Simulation Using MATLAB

Steven P. Levitan, Jose A. Martinez and Donald M. Chiarulli
University of Pittsburgh, Pittsburgh, PA

Abstract

Design exploration is most often accomplished in a design, analysis, evaluate loop which is limited by the speed and quality of the simulation tools used for analysis and evaluation. For new technologies these simulations are often ad-hoc and incompatible with traditional simulation environments. Therefore, it is challenging to model systems composed of novel devices together with conventional devices. Similarly, development of new methodologies for model extraction, reduction, and abstraction are often hampered by the implementation overhead required to apply these methodologies in an environment compatible with existing models.

To address both these needs, we have developed a flexible MATLAB simulation platform that supports SPICE netlists in conjunction with abstract device models for compatible simulations as well as enabling comparisons between different extraction, modeling and simulation techniques with a common baseline environment.

1 Introduction

Increased design productivity is predicated on the development of new CAD tools for exploring larger design spaces more quickly and more thoroughly than can be achieved with current techniques. This design exploration is most often accomplished in a “design”, “simulate”, “analysis” loop which is limited by both the speed and quality of the simulation tools used for analysis and evaluation. For many designs, well known models and design abstractions allow for the accurate analysis of both analog and digital circuits using circuit simulation tools such as SPICE [1]. It is often possible to use higher level abstractions and faster simulators for timing and power estimations of large digital systems.

However, for new technologies the simulation of the behavior of new devices and components are often ad-hoc and incompatible with traditional simulation environments. Device model development is often done using physics based models implemented directly in a programming language or with mathematical modeling tools such as MATLAB [2]. Therefore, it is challenging to model systems composed of novel devices together with conventional devices (e.g., CMOS) in a single evaluation environment. There is an increasing need to evaluate these new devices, which are often based on new technologies, or hybrids of existing technologies, and to simulate circuits and systems that include both new and traditional devices.

Not only do we need to provide a methodology for these multi-technology simulations, but also, there is a need to evaluate new simulation techniques and new model extraction and abstraction techniques, such as reduced-order modeling. It is easier to evaluate the quality of these new modeling methodologies and simulation algorithms if they can be easily integrated into an environment where standard circuit netlists can be parsed and simulation results can be compared to existing tools. However, integrating these new ideas into open source versions of SPICE or other simulators is a time consuming and needlessly frustrating exercise. On the other hand, MATLAB gives a very flexible environment and is currently in widespread use for model development. However, to our knowledge, there are currently no non-linear circuit solvers, or analog solvers, built into MATLAB and no way to parse circuit netlists (e.g., SPICE “decks”) into MATLAB.

In this paper we present our MATLAB platform for the simulation of novel devices in the context of traditional CMOS circuit netlists. The environment also provides a means to perform comparisons between different extraction, modeling and simulation techniques within a common modeling and simulation tool.

2 Approach

Our work, to date, has resulted in a Perl[3] based SPICE netlist parser and translator that generates MATLAB script files and supports such useful SPICE constructs as sub-circuits, input sources, and transistor model parameters. The script is used as input to our MATLAB non-linear circuit solver. The solver is based on the same non-linear iterative stamp based technique developed for SPICE, and currently supports adaptive time-stepping, initial condition convergence, as well as Backwards Euler and trapezoidal integration methods. Generic SPICE options are provided such as RELTOL, ABSTOL, VNTOL, LVLTIM, and GMIN. The solver currently has hard coded models for resistors, inductors, capacitors, diodes and default MOS transistors. Support is also provided for custom MOS models as well as generic multi-port non-linear electronic devices.

Given that solver is written in MATLAB, the environment provides the ability to dump and load the internal data structures, and thus generate snapshots of
the system state, for interaction with external circuit analysis and optimization tools.

It is important to note that these tools are in no way replacements for SPICE or any other simulation tool developed primarily for fast simulation of analog circuits. Rather, these programs were developed to support our own research in non-linear circuit simulation and model order reduction. Along the way, we realized it would be useful to be able to use SPICE netlists for our tests. Therefore, we have written these support tools to translate a simple SPICE netlist into MATLAB scripts files along with some supporting MATLAB function files and a wrapper function. The wrapper function ties these all together and invokes the non-linear circuit solver. Our goal is to make these tools available for other researchers who are doing similar research using MATLAB to allow them to work with common netlist formats and a stable simulation environment.

3 Parsing SPICE netlists

Starting with the goal to simulate a variety of CMOS circuits without having to hand code their netlists into MATLAB data structures, we developed a netlist parser. The Perl function parses a SPICE netlist and generates a set of MATLAB functions callable by the solver, as well as a MATLAB script file representing the netlist itself. The parser is modular and could be used in other environments to parse SPICE files into other formats.

The parser is based on the Perl package, Parse::RecDescent [4]. The basic operation of the parser is simply to take the R, L, C, and M, element statements of the netlist and convert them into function calls to add elements into the internal simulator data structures. Element parameters with SPICE scale modifiers (e.g., K, U, P, and F) are supported. For MOS transistors, length, width, and model names are understood. Voltage and current sources are mapped to MATLAB functions which return voltage or current values as a function of time. MOS “model cards” are converted into MATLAB functions which are invoked by the simulator to set device parameters prior to circuit evaluation.

To make it easier to parse common circuits, file includes, line continuations and comments are supported. Nested sub-circuits are also supported, which is probably the most useful aspect of the whole parser.

The output of the parser is a MATLAB script file. The script file is evaluated by the simulator resulting in the creation of the internal data structures for the solver including the netlist structures for the voltage and current sources.

4 Simulation Engine

A typical analog simulator can be characterized by the way it performs three major tasks: nodal analysis, linearization of nonlinear elements, and integration of time dependent components, as it moves the system through a series of discrete points in time.

Nodal Analysis - The primary task of any analog solver is to convert the system under analysis into a linear representation that is valid at each evaluation time, thus capturing the behavior of the system. From that representation, any linear analysis method can be applied to derive the current state of the system and, in doing so, encapsulate the response in physically identifiable variables (e.g., current, voltage, force). In the majority of analog solvers in use, a variation of nodal analysis is used to describe the system [5]. Under this methodology, characteristic points in the circuit which correspond to concentrated values of a variable of interest are identified and principles of conservation of flow and potential are then used to describe the governing relationships (e.g., electrical current and voltage, and corresponding Kirchoff’s laws for electrical circuits).

In our current analog solver implementation, Modified Nodal Analysis (MNA) is chosen to create a mathematical representation for electrical netlists, as shown in figure 1. In this expression, M corresponds to the memory matrix of the system, also called the susceptance matrix, R is the conductance matrix, C is the vector of state variables, B is a connectivity matrix, and u is the excitation vector, and the y vector that contains the variables to evaluate. The linear elements can be directly mapped to this representation, but the nonlinear elements need to first undergo a further transformation.

Linearization of nonlinear elements - Nonlinear devices in the system are replaced by equivalent stamps (also called templates) that are simply the local nodal matrix representation of the linearized element at the current operating point. Because of the nonlinear behavior of these elements, an iteration loop is required to reach a convergence state. This involves finding a state for the current evaluation time that satisfies all the elements in the system.

Integration of time dependent components - For time dependent elements in the system, such as energy storage devices (e.g., capacitors and inductors) an
integration loop is required. A linearization of these elements is provided by using one of several integration techniques and the error in their application is estimated with an associated error bound algorithm. The convergence state reached in the inner loop is validated to meet the error tolerance that has been chosen for the evaluation.

These three core tasks are used repeatedly by the simulator to perform its job of evaluating the operation of the system over time.

4.1 Operating point and transient simulation

The minimal configuration for an analog solver involves two major stages: Initial operating (IO) point evaluation (also known as DC evaluation) and transient evaluation. Transient evaluation requires a valid state for the system to start from, consequently is usually preceded by the IO step.

Initial operating point evaluation - The first major stage for an analog solver is to find a valid operation point for the system at the requested initial time. Several strategies have been developed and implemented through the commercial development of practical circuit solvers [6]. It is important to remark that in many situations, this is a crucial step and sometimes the most difficult bottleneck point in the simulation. For this project, we have chosen to implement several strategies to provide a highly reliable IO step that is also efficient in terms of computation time. The algorithm uses three techniques.

- The use of GMIN (minimum conductance) as a leakage parameter for the initial estimation of an operating point. These temporary leakage conductances are added between each node and the reference node (GND). This strategy forces all loops in the electrical circuit to be closed, avoiding the possible situation of unconnected (i.e., floating) nodes, which are usually found when using nonlinear circuit elements during IC (initial condition) evaluation. Floating nodes can cause an ill-conditioned transfer matrix for the system, and in doing so create convergence problems.
- A first pass “cold evaluation” attempt. This is a simple evaluation of the circuit that does not consider temporal dependencies (derivatives). Therefore, the storage elements in the circuit (C, L) are considered in steady state. Capacitors act as voltage sources and inductors as current sources with initial values corresponding to the IC values specified in the netlist, or zero if not specified.
- A second pass, source stepping algorithm. This is a stable and reliable algorithm where the circuit is considered initially not energized. Then, a progressive step by step increase in the value of individual power (independent) sources is performed. The final value of each stepping progression is the DC value of the source in the netlist. When all of the sources are completely stepped, a final operating point is achieved. While this is not the only possible operating point, if it is reached, it is a valid one, which is the main goal.

Even though we are reaching for a DC operating point, we must consider the time dependencies of the system. Therefore, this algorithm tries to achieve convergence in the nonlinear evaluation by first using a simple time stepping algorithm, and in case of failure, reinitiates the evaluation using local truncation based error (LTE) strategy for the dynamic control of timesteps [7]. This combination has proven to be both reliable and efficient.

Transient Analysis - In this stage, the goal is to successively move the system through the given time period, generating its proper response within the allowed error tolerance at each advancing timestep. Starting from the operating point found in the previous stage, an iterative process is initiated to obtain the desired response.

The main purpose of this stage is the convergence algorithm, which, for this implementation, is based in an iterative gradient method. The main kernel of this algorithm is shown in figure 2. We briefly describe each step.

1. A validation of state information, operating point and input and output requirements is performed.
2. The generation of stamps (i.e., templates) for storage elements is accomplished in this step. These stamps correspond to the linearization of each element based on the integration method selected.
3. This the top of the main convergence loop. Nonlinear elements stamps are created corresponding to the operating point (at time \( t_n \)). This is followed by an integration of these linear stamps into the MNA representation of the circuit.
4. A linear solution is generated corresponding to the predicted new operating point, for time \( t_{n+1} \).
5. A check for convergence is performed. If the solution is not convergent, (6) decrease the timestep previously used, based on the dynamic control algorithm in use.
6. A dynamic control of the time step is performed. During this step, even if the system is convergent, the evaluation point can be rejected based on an unacceptable estimated error. On the other hand, if the evaluation is accepted, generate the next timestep to use, possibly relaxing the current time interval.
8. If the end of the interval is reached terminate the algorithm. If not, use the proposed next timestep calculated above.

5 Examples

Figures 3 (a) and (b) show $V_{ds}$ vs. $I_{ds}$ sweeps for two NMOS transistor models in the simulator. Figure 3(a) shows an implementation of the SPICE NMOS “long channel” model from [8]. This is the default model in the simulator. Figure 3(b) shows the “unified short channel” model from [9] which has also been implemented. Note the figures are not on the same scale. Figure 3(c) shows the output waveforms from our tool at three taps on a nine stage digital ring oscillator built from CMOS inverters, with each stage having a capacitive load.

Figure 4 shows an NMOS astable multivibrator simulated in HSPICE and in our MATLAB simulation environment. The HSPICE simulations used level 1 parameterized MOS models. The spice file was directly parsed into the MATLAB format, including the model parameters and run in the simulator. As shown in figures 4(b) and 4(c), both HSPICE and our simulator found initial conditions and simulated the circuit breaking into oscillation. However, in this case different (but both valid) initial conditions were determined by the two programs. Figures 4(b) and 4(c) also show that the period of the two simulations differ by about 10%. This is probably due to the lack of a parasitic companion capacitance in our current implementation of the MOS model.

Figure 5 shows a simple CMOS differential amplifier simulated in HSPICE (level 1) model and the same circuit simulated in our tool. It is important to note that the solver needed no help finding the initial operating point, and once it was established, the simulation kernel was quite efficient. On average, the solver took three iterations to converge at each timestep, compared to an average of two iterations per time-step for HSPICE.

6 Generic Nonlinear Models

In support of new device models, we provide a generic user defined nonlinear device stamp. The device model can be a stand-alone file with a MATLAB function that represents the custom behavior of a device. The external view of the model should be a multi-port nodal description. The function interface to the simulator need only provide the output nodal current vector, a Jacobian matrix and an optional Hessian tensor for a given input vector of the device’s node potentials. Device instantiation is by adding an element into the circuit description with the appropriate nets in order, and the name of the function as the “device model”. The function will be called as needed by the simulator for both initial operating point and transient analysis.

7 Summary and Conclusions

The result of this work is a MATLAB platform for the investigation of both new models and new simulation techniques compatible with existing SPICE models of devices and netlists of circuits and systems. This will enable device designers to develop MATLAB models and perform circuit simulation of devices based on new circuit fabrication techniques and new material systems including nano-electronics, molecular electronics, organic semiconductors, carbon nanotube devices, etc. This same environment will enable the evaluation of new methodologies for model extraction, abstraction, reduction, and simulation.

8 Acknowledgments

This work was supported, in part, by the National Science Foundation grant CCF-0541150.
9 References


Figure 3: Ids sweeps using (a) Long Chanel [8] and (b) Short Channel [9] models (Note: scales not the same) (c) 3 nodes of a 9 stage CMOS digital ring oscillator in MATLAB

Figure 4: (a) Astable Multivibrator, (b) HSPICE, (c) MATLAB simulations (same scale)

Figure 5: (a) CMOS Differential Amplifier, (b) HSPICE, (c) MATLAB simulations (same scale)