Code-level modeling in XSPICE

F. L. Cox III, W. B. Kuhn, J. P. Murray and S. D. Tynor

From the
Proceedings of the IEEE International Symposium on
Circuits and Systems, 1992 (ISCAS '92),
Volume 2 , pp. 871-874,
10-13 May 1992.

CODE-LEVEL MODELING IN XSPICE

Fred L. Cox III, William B. Kuhn, Jeffrey P. Murray, and Stephen D. Tynor

Computer Science and Information Technology Laboratory
Georgia Tech Research Institute
Georgia Institute of Technology, Atlanta, GA 30332

ABSTRACT

This paper describes code-level modeling in XSPICE, an extended version of the SPICE3 simulator from the University of California at Berkeley. XSPICE extends SPICE3's capabilities to allow efficient simulation of mixed-signal (analog/digital) circuits and systems. XSPICE's code-level modeling approach allows new models to be easily added to the SPICE3 core, providing a practical alternative to traditional macro-modeling techniques. Addition of event-driven simulation extends SPICE3 to include 12-state digital modeling. User-defined data types allow simulation of designs such as sampled-data filters. A library of over 40 predefined code models has been developed, covering analog, digital, and hybrid devices and functions. The complete XSPICE simulator has been integrated with the Mentor Graphics MSPICE™ CAE environment, and provides a general interprocess communications interface for connection to other CAE system software.

INTRODUCTION

The ability to simulate analog and mixed-signal circuits from the discrete and IC levels through board and system levels can prove valuable in many engineering problems. Examples include top-down design of complex electronic systems, and determining normal, faulted, and tolerance behavior of existing circuits. Unfortunately, traditional analog simulators such as SPICE [1] are targeted primarily at low-level IC design and provide little support for board-level and system-level domains.

The SPICE program's set of built-in device models is limited to discrete components (resistors, diodes, transistors, etc.) and to simple linear and polynomial controlled sources [1,2]. To allow SPICE to simulate board-level circuits, macro-modeling approaches have been developed in which discrete devices and controlled sources are combined to create more efficient models of ICs such as operational amplifiers [3,4,5]. Some simulators now provide higher level primitives such as limiters, multipliers, and user-defined equations [6] to better support the macro-modeling approach, and a few allow users to extend the simulator's set of primitives by coding models in a general-purpose programming language and linking them with the simulator core [4,7]. This "code modeling" approach has the potential to allow users to describe arbitrarily complex functions and behavior, freeing them from a restricted set of devices or functions, and from development time and execution time inefficiencies inherent in the traditional macro-modeling approach.

Nevertheless, the utility of the code modeling approach has not been widely recognized. We believe this is due largely to the limited provisions for code modeling in available simulators and to a lack of programming support. Our work has been aimed at making code modeling a viable alternative to traditional macro-modeling technology by providing usable code modeling tools in a modern SPICE-based simulation environment. This code-level modeling support extends SPICE3 to the board-level and system-level domains.

OVERVIEW

This paper describes the modeling support and associated components of XSPICE, a new board and system-level simulation tool developed at the Georgia Tech Research Institute. XSPICE is built around the SPICE3 program from the University of California at Berkeley [1], inheriting the major features of SPICE simulation while extending the program to work in new domains through the following new capabilities.

- o Support for adding "code models" written in the C programming language.
- o Code model development and linking tools.
- o Extensions to the SPICE circuit description syntax.
- o Embedded event-driven simulation.
- o Full 12-state digital modeling.
- o Support for simulating with arbitrary, user-defined, node types.
- o An extensive library of analog and digital functional-level code models.
- o Enhancements to SPICE algorithms for board and system-level simulation.
- o An interprocess communication interface for connection to CAE system software.

A top-level diagram of XSPICE is shown in Figure 1. A Code Model Toolkit assists the user in writing, compiling, and linking new models with the simulator. This toolkit works with additions and modifications made to the SPICE3 core to tell the simulator how to parse the extended circuit description syntax and how to call the C code that defines the model's behavior. The toolkit also provides support for introducing new event-driven data types (user-defined node types) and for linking any number of code models and user-defined node types with the core to produce new simulator executables. Preprocessing and compile times for a typical code model or user-defined node type are approximately 30 seconds, and a new simulator can be linked in as little as 60 seconds (using an HP Apollo 9000 Series 400 workstation).

0-7803-0593-0/92 $3.00 1992 IEEE

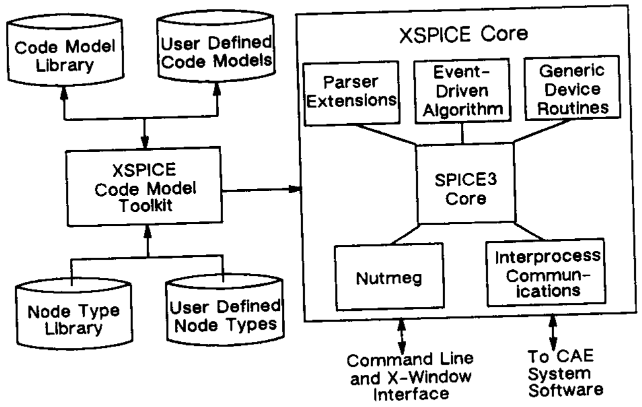

Figure 1. XSPICE Top-Level Diagram.

THE CODE MODEL TOOLKIT

The Code Model Toolkit shown in Figure 1 assists users in adding new models and node types to the simulator core, while insulating them from the underlying algorithms and data structures of SPICE3 and XSPICE. Creating new models and adding them to the simulator involves six steps, including creating directories for the models and for the simulator executable, defining the models in Interface Specification and C Function files, and building the models and simulator with the UNIX™ 'make' utility. These steps are easily learned and require only a few minutes to perform once the C code for the model has been developed. Creating a new, user-defined, node type follows similar steps and can be accomplished in a similar period of time.

The Interface Specification File

A template for the Interface Specification file is created automatically when the user creates a model directory. The user edits this file to define the model's inputs, outputs, and parameters. This capability allows a high degree of flexibility to the user in describing various aspects of models. An example Interface Specification file is shown in Figure 2.

```
NAME_TABLE:
C_Function_Name:      cm_gain
Spice_Model_Name:     gain
Description:          "A simple gain block"

PORT_TABLE:
Port_Name:            in                       out
Description:          "input"                  "output"
Direction:            in                       out
Default_Type:         v                        v
Allowed_Types:        [v,vd,i,id,vnam]         [v,vd,i,id]
Vector:               no                       no
Vector_Bounds:        -                        -
Null_Allowed:         no                       no

PARAMETER_TABLE:
Parameter_Name:    in_offset         gain        out_offset
Description:       "input offset"    "gain"      "output offset"
Data_Type:         real              real        real
Default_Value:     0.0               1.0         0.0
Limits:            -                 -           -
Vector:            no                no          no
Vector_Bounds:     -                 -           -
Null_Allowed:      yes               yes         yes
```

Figure 2. Example Interface Specification.

Since the file is in a simple tabular text format, it also serves as documentation on the model's use.

The C Function File

A template file for the C function is also automatically created when the model directory is created. The user edits this file to describe the functional behavior of the model. The Code Model Toolkit provides the user with a number of macros and functions to simplify the process of accessing information needed by the model. Examples of these are listed below with brief explanations. This is by no means an exhaustive list but represents the nature of support available to modelers.

INPUT( )	Accesses the value of the named input.
OUTPUT( )	Used to assign computed value to the named output.
PARAM( )	Accesses the value of the named model parameter.
ANALYSIS	Provides type of analysis (DC, AC, TRANSIENT).
TIME	Accesses the current timepoint value.

cm_analog_alloc( )	Create storage to hold data between calls.
cm_analog_integrate( )	Integrate the specified quantity.
cm_event_queue( )	Queue an event time at which to call the model.

The macros insulate the user from detailed data structure definitions used to pass input, output, and parameter data to the function and allow this data to be referenced in the C function by the names supplied in the Interface Specification. In addition, they allow modifications to be made to the parameter passing mechanism in future versions of the simulator without invalidating existing models. An example C function for a simple, gain-block model is shown in Figure 3.

```
void cm_gain(ARGS)
{
    if((ANALYSIS == DC) || (ANALYSIS == TRANSIENT)) {
        OUTPUT(out) = PARAM(out_offset) + PARAM(gain) *
                      (INPUT(in) + PARAM(in_offset));
        PARTIAL(out,in) = PARAM(gain);
    }
    else {
        AC_GAIN(out,in).real = PARAM(gain);
        AC_GAIN(out,in).imag = 0.0;
    }
}
```

Figure 3. Example C Function

CIRCUIT DESCRIPTION SYNTAX EXTENSIONS

XSPICE provides much of its flexibility and power through its extensions to the SPICE circuit description syntax. These extensions include:

- o Support for any number of inputs and outputs (ports) on a model.
- o Scalar and vector ports.
- o Ground referenced and differential ports.
- o Bidirectional ports for modeling I-V relationships.
- o 12 pre-defined port types (voltage, current, conductance, 12-state digital logic, etc.).
- o Port-type overrides to allow a single model to work with voltages, currents, etc.
- o User-definable port types.
- o Logic inversion support for event-driven ports.

- o Use of the 'null' keyword to indicate a port is unused.
- o Parameters of type real, integer, complex, string, or Boolean.
- o Scalar and vector parameters.
- o Parameter defaults and constraint checking.

References to code models in a SPICE deck follow the general form used by SPICE for semiconductors. An instance name for a code model must begin with the letter 'a' to distinguish it from other SPICE devices. The instance name is followed by a list of connections and then by a user-chosen model name referencing a .model card. The name of the code model and the model parameters are supplied on the .model card as in standard SPICE.

This syntax is automatically parsed by the XSPICE simulator based on the definitions of ports, port types, and parameters supplied in each code model's Interface Specification. Special characters such as [ ] < > % and ~ are used to delimit vectors, complex numbers, port-type overrides, and logic inversion in connection lists and parameter lists. The following examples illustrate the appearance of the XSPICE circuit description syntax for a zener diode code model, and for a Nand gate code model with four inputs, two of which are inverted:

```
azener  1 2  1n751a
.model 1n751a zener (v_breakdown=5.1 i_breakdown=20mA
+  r_breakdown=8.0 i_reverse=400nA i_sat=2.37e-16
+  n_forward=1.0)

anand  [addr_1 addr_2  ~addr_3  ~addr_4] sel1  nand_gate
.model nand_gate d_nand (rise_delay=1.5ns fall_delay=0.8ns)
```

THE CODE MODEL LIBRARY

A library of over 40 predefined code models has been developed for XSPICE. These are grouped into analog, digital, and hybrid models, depending on the types of data with which they operate.

The predefined analog code models operate with the standard analog data (voltages and currents) of SPICE. Examples of models in this category include Piecewise-Linear Controlled Sources, Zener Diodes, S-Domain Transfer Functions, Magnetic Cores, and Controlled Oscillators.

The predefined digital code models manipulate 12-state digital data, and are simulated by the embedded, event-driven, simulator algorithm. Examples of digital models include Gates, Flip-Flops, Digital Sources, State Machines, and RAMs.

The predefined hybrid code models operate with SPICE voltages and currents and with the event-driven simulator, providing a link between analog and event-driven algorithms during simulation.

EVENT-DRIVEN SIMULATION ALGORITHM

Modern circuits and systems often contain both analog and digital components. Traditional analog simulation methods such as those used in SPICE are generally inefficient for simulating such mixed-signal designs. Simulator developers are therefore beginning to integrate analog simulation with event-driven simulation.

Event-driven simulation in XSPICE is implemented for DC and Transient analysis. Event data and event times are processed independently of, but coordinated with SPICE3's iterations and timestepping. The conversion between event-driven data and SPICE analog voltages and currents is handled by hybrid code models that work in both domains. Hybrid code models coordinate event timing with analog timesteps through calls to functions in the XSPICE C function support library. These calls set breakpoints in the analog simulation timestepping and queue events in the event-driven simulation. The underlying XSPICE code handles fixup of output states and event queues whenever the analog algorithm rejects a timepoint and decreases the time delta to address numerical integration local truncation error limits or convergence problems.

USER-DEFINED NODE TYPES

The event-driven algorithm used in XSPICE was implemented independently of the data structure representation used for digital modeling to allow the same algorithm to be used with arbitrary user-definable data structures. Code models receive generic pointers to these data structures and cast the pointers to the desired types. Hence, the details of the data do not need to be known by the event-driven algorithm. To add a new type for event-driven simulation, the user creates a set of functions that perform all the primitive operations required by the event-driven algorithm, such as structure creation, initialization, copying, and comparison.

Creation of functions for a new data type is relatively easy. For example, creation of the full set of functions for the 12-state digital data type required approximately 3 hours of design and coding. Creation of functions for simple real and integer data types required only about 30 minutes each.

PARTIAL DERIVATIVES

One of the problems with adding analog models to SPICE-type simulators at the code level is the need for partial derivatives of outputs with respect to inputs. Partial derivatives are required by the simulator to solve non-linear, simultaneous equations through Newton-Raphson iteration[1].

The XSPICE user has the option of coding partial derivatives directly, or of requesting XSPICE to automatically compute them through a call to function cm_analog_auto_partial(). In the latter case, XSPICE will call the model N additional times, where N is the number of analog inputs. At each additional call, a single input is varied by a small amount, and the partial of each output with respect to that input is approximated by divided differences. The amount by which the input is varied is equal to the convergence tolerance used by the analog simulation algorithm.

SIMULATION EXAMPLE

An example illustrating the use of XSPICE in top-down, system-level design is shown in Figure 4. The system is a simple MIDI (Musical Instrument Digital Interface) synthesizer. Simulation of the synthesizer involves three different types of data. Digital data is used on the left for input of MIDI note numbers and note on/off control and for simulating a numerically controlled oscillator and frequency divider. Real-valued data is used in the center to simulate sampled-data filters used in converting the rectangular waveform from the digital divider into the desired waveshape. The output of the sampled-data filter is then converted to normal SPICE voltage and current data for low-pass filtering at the circuit level. Plots of the frequency divider's output, note on/off bit, filter output, and opamp output are shown in Figure 5.

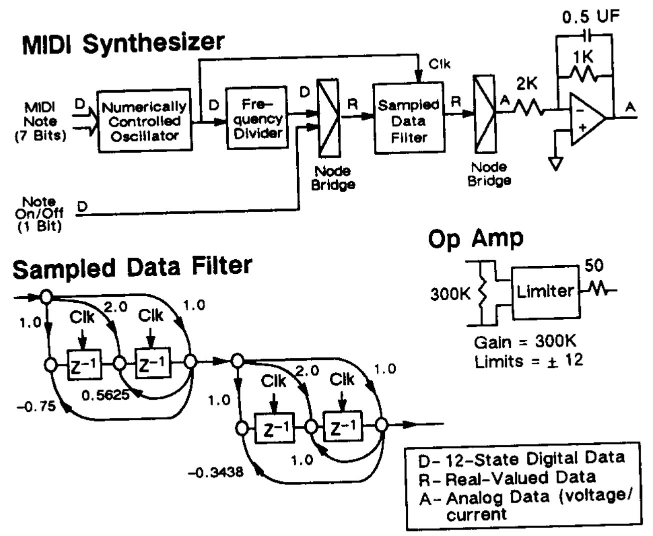

Figure 4. System-Level Simulation Example.

The development of this simulation required approximately one-day's work, including developing 5 new code models. Simulation run time was under 1 minute on an HP Apollo 9000 Series 400.

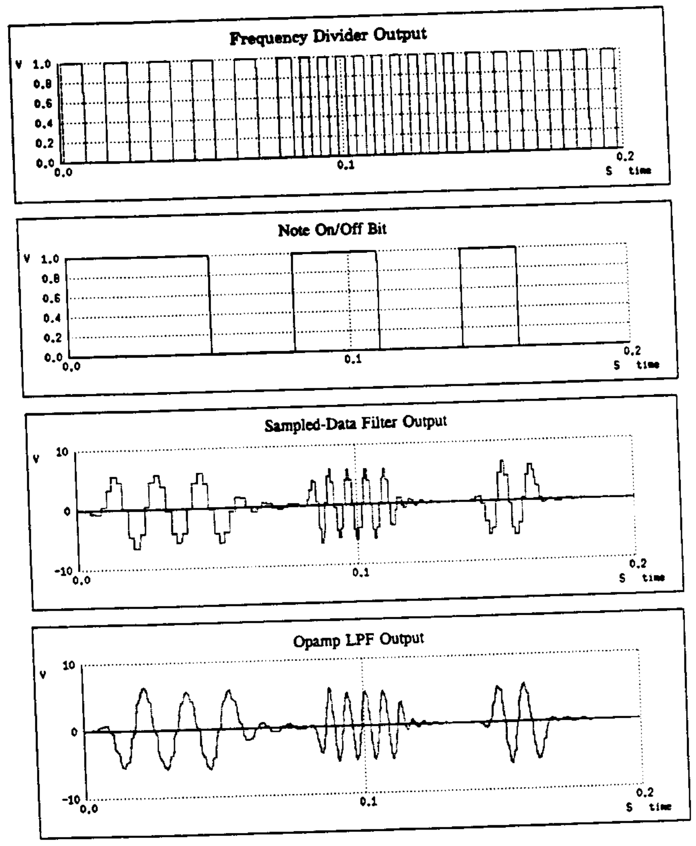

Figure 5. Simulation Example Output Waveforms.

APPLICATIONS

The XSPICE simulator is currently being used at Georgia Tech and additional sites to analyze board-level, avionics circuits. Efficient analysis of such circuits is made possible by XSPICE's code-level modeling and embedded event-driven simulation. These features allow normal and faulted behavior of the circuits to be analyzed and make transient analysis, Monte-Carlo tolerance simulations practical. In another application, the XSPICE simulator is being used to model the biological processes found in wastewater treatment plants. Code models of reactors, separators, and valves provide the building blocks for simulating, evaluating, and optimizing design alternatives.

We anticipate that XSPICE will find additional applications in engineering education and in design at the IC, circuit board, and system levels. The ability to describe arbitrarily complex behavior in a widely used programming language makes new modeling techniques accessible to the general SPICE user community.

ACKNOWLEDGEMENTS

The XSPICE simulator is based on the SPICE3 program developed by the Electronic Research Laboratory, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley. The design and development of XSPICE was performed as a part of projects funded by the U.S. Air Force (ASD and WR-ALC). Additional thanks are due Harry W. Li and Stefan P. Roth for their many contributions to the success of the project.

REFERENCES

[1] Quarles, Thomas L., "SPICE3 Version 3C1 User's Guide," Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California, April 1989.

[2] Vladimirescu, A., K. Zhang, A. R. Newton, D. O. Pederson, A. Sangiovanni-Vincentelli, "SPICE Version 2G User's Guide," Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California, August 1981.

[3] Boyle, G. R., B. M. Cohn, D. O. Pederson, "Macromodeling of Integrated Circuit Operational Amplifiers," IEEE Journal of Solid State Circuits, Vol. SC-9, No. 6, December 1974.

[4] Epler, B.,"SPICE2 Application Notes for Dependent Sources," IEEE Circuits and Devices Magazine, September 1987.

[5] Sitkowski, M., "The Macro-Modeling of Logic Functions for the SPICE simulator," IEEE Circuits and Devices Magazine, September 1990.

[6] Hageman, S., "Behavioral Modeling and PSpice Simulate SMPS Control Loops," PCIM Magazine, April/May 1990.

[7] Bowers, J. C., R. S. Vogelsong, "Computer Aided Design for Power Electronics," IEEE Applied Power Electronics Conference and Exposition, New Orleans, LA, 1986.

[8] "Saber Cuts SPICE Out of Analog Simulation," Electronics, pp. 80-82, October 30, 1986.

Note 1. The Saber™ simulator from Analogy Inc. [8] eliminates the need for explicit coding of partial derivatives but requires the selection of sample points for piecewise linearization - a technique which can be difficult for functions with more than a single input.

Note 2. MSPICE is a trademark of Mentor Graphics Corporation. UNIX is a trademark of AT&T. Saber is a trademark of Analogy Inc.