32-nd Midwest Symposium on Circuits and Systems (MSCAS'89); Champaign, IL, August 14-16, 1989, pp.985-988.

HIGHER-LEVEL MODELLING EXTENSIONS TO SPICE-LIKE CIRCUIT SIMULATORS

W.M. Zuberek

Department of Computer Science, Memorial University
St. John's, NL, Canada A1C-5S7

A b s t r a c t

Three different but complementary approaches to higher-level specification of analog circuits are proposed as a set of extensions to SPICE-like circuit simulators. They include table-driven, formula-driven and program-driven generalized characteristics of (nonlinear) circuit elements as well as whole "blocks". Examples of SPICE-PAC enhancements are used as an illustration of these approaches.

1. INTRODUCTION

Most designers have accepted the need for circuit simulation before the expensive and time-consuming fabrication of their designs takes place. Even with designs of modest size, an accurate simulation provides invaluable insight into the performance of the product. However, the existing trend in integrated circuit design towards increasing device densities and shrinking circuit geometries results in designs of increasing complexities. This – in turn – increases the simulation time and memory requirements, and as they grow more than linearly with the size of the simulated circuit, it is practically impossible to satisfy these requirements for integrated circuits using classical circuit simulators. Several techniques have been proposed to counterbalance these increasing requirements [DeLS,DiNaV,FRS,WSSN], but using "higher-level abstraction" wherever possible remains the simplest and the most effective solution to this complexity problem.

The notion of "abstraction" in this context means a method to replace an object by a simplified one that only defines the interaction of the object with its environment, while deleting the internal organization details of the object [Niess]; for example, the whole comparator module can be replaced by a single "block" with input-output characteristics equivalent to those of the original design. The virtue of abstraction is data reduction, sometimes by one or more orders of magnitude. For very large systems, one level of abstraction may not suffice, it has to be applied a number of times in succession creating "hierarchical abstraction".

In digital systems, the hierarchy of abstraction seems to be well established [Niess,ORour,Whar], with the behavioral level at the top, followed by the functional, gate, and circuit levels to device or even process level at the bottom. It is characteristic that "lower" levels of abstraction are well defined, with clear boundaries between levels, while the definitions of behavioral and functional levels are rather vague and ambiguous. Perhaps the "best" characterization of behavioral specification is a "black box" description of a component [Davis] which indicates how the information leaving a component is related to the information that entered it. That is [Whar], behavioral description characterizes the relationships among the activities of the ports of a node, without reference to the internal logic or physical structure of that node; when described behaviorally, a node has no contents. Behavioral specification is thus an abstract mathematical relationship between activities at the input and output terminals of a component. A variety of techniques have been used in the past to describe behavior, including simple rules for mapping inputs to outputs, Petri nets to represent concurrency and parallelism, and unrestricted fragments of code; various combinations of these three have also been explored [Davis]. More recently, high-level languages have been proposed to specify behavior of systems [Bart]; since such languages deal with basically mathematical (abstract) specifications, they share many features with "typical" high-level algorithmic languages (for example, ADA [Bart]).

The functional description shows how the structure and components of a system contribute to its overall function [Kuip]; the functional description of a system should make explicit not only what behaviors are possible for a system, but also why. A typical approach to functional description of a system is to combine functional descriptions of system's components according to the structure of a system; for example, a functional description of a logic network is a (logic) equation obtained by a composition of network's gate functions.

For analog systems the situation is similar; only the bottom levels, i.e., the process, device and circuit levels, are clearly identified [Getr]. Quite often "macromodelling" is considered as the functional level, while specification at the level of differential equations corresponds to the behavioral level.

For the purpose of this paper, all specification (or description) capabilities that are "above" the circuit level are generally referred to as "higher levels" of abstraction.

Three different but complementary approaches to higher-level simulation are proposed in this paper as extensions to the SPICE program, the most popular circuit simulator which has become a "de facto" standard in circuit simulation. The extension have been implemented in the SPICE-PAC simulation package, a simulation tool that is upward compatible with SPICE; it means that SPICE-PAC accepts the same circuit descriptions and provides the same set of circuit analyses as SPICE, but it also offers a number of enhancements not available in the original SPICE-like simu-

lators [Zub].

The proposed approaches are: (i) table-driven (or numerical); very efficient (if a suitable organization is used) but rather inflexible, (ii) formula-driven (or analytic) - quite flexible but less efficient if complex evaluations are involved, and (iii) program-driven, in which user-supplied routines or interfaces to other software tools perform the required evaluations.

These three approaches are discussed in the following sections in the context of SPICE-like simulators, with some implementation detailes and examples provided by the SPICEPAC package.

2. TABLE-DRIVEN SIMULATION

The characteristics of nonlinear elements (capacitors, inductors, dependent sources) as well as "input-output" characteristics of modules (or "subcircuits") can be specified by (multidimensional) tables of numerical data with an associated interpolating method. Typical table-driven description follows the following syntax (with some small differences for different classes of elements):

```
name node+ node- itp(n) arg1 arg2 ... argn x1,x2,...
```

where `name` is the unique name of the element; `node+` and `node-` are element terminals; `itp` indicates the interpolation method:

`itp`:	value:	derivative:
`PWL` or `PWL1`	piecewise linear	approximated
`PWL2`	piecewise linear	piecewise linear
`PWQ` or `PWQ1`	piecewise quadratic	approximated
`PWQ2`	piecewise quadratic	piecewise quadratic
`PWC` or `PWC1`	piecewise cubic	approximated
`PWC2`	piecewise cubic	piecewise cubic

`n` is the number of arguments (or controlling voltages and/or currents); `arg1, arg2, ... argn` are the arguments (in the SPICE sense, i.e., each `arg` is either a pair of nodes that determine the controlling voltage, or a voltage source name that indicates the controlling current flowing through this source); finally, `x1,x2,...` are the consecutive numerical data elements describing the characteristic of `name`.

In the following example, capacitance C (of a nonlinear capacitor or a "capacitive block") is an exponential function of $Vcap$, the voltage over the element, $C = a*(1-b*e^{-Vcap})$, with $a = 100pF$ and $b = 0.5$, and this characteristic is given as a table-driven description with a sequence of ordered data triples composed of the controlling voltage, the corresponding charge and the capacitance:

```
Cexp  4 0   PWQ2(1) 4 0    (0.00 0.000E+00 5.000E-11,
+ 0.01 4.520E-13 5.050E-11, 0.03 1.010E-12 5.148E-11,
+ 0.05 2.039E-12 5.244E-11, 0.07 3.088E-12 5.338E-11,
+ 0.10 5.242E-12 5.476E-11, 0.30 1.704E-11 6.296E-11,
+ 0.50 3.033E-11 6.967E-11, 0.70 4.483E-11 7.517E-11,
+ 1.00 6.839E-11 8.161E-11, 2.00 1.568E-10 9.323E-11,
+ 3.00 2.525E-10 9.751E-11, 5.00 4.503E-10 9.966E-11)
```

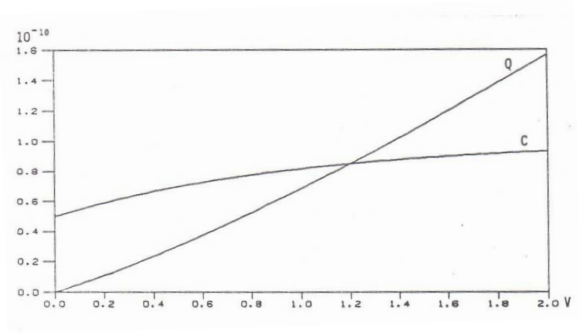

Fig.1. Capacitance C and charge Q of `Cexp`.

Since the capacitance values are "secondary" information in this table (capacitance is equal to the derivative of the charge with respect to the controlling voltage), a slightly less accurate results can be obtained using only the values of charge a numerical approximation of capacitance:

```
Cexp  4 0  PWQ1(1) 4 0   (0.00 0.000E+00,
+  0.01 4.520E-13, 0.03 1.010E-12, 0.05 2.039E-12,
+  0.07 3.088E-12, 0.10 5.242E-12, 0.30 1.704E-11,
+  0.50 3.033E-11, 0.70 4.483E-11, 1.00 6.839E-11,
+  2.00 1.568E-10, 3.00 2.525E-10, 5.00 4.503E-10)
```

Fig.1 shows the charge and capacitance as functions of the voltage over the capacitor `Cexp`.

3. FORMULA-DRIVEN SIMULATION

The characteristics of (nonlinear) elements can also be specified by algebraic formulas with controlling voltages and/or currents as parameters. During initial processing of circuit descriptions, the formulas are translated into an intermediate form, which is evaluated whenever the element attributes are needed for formulation of circuit equations. The syntax for formula-driven elements uses `ARG(n)` to indicate the number `n` of "arguments", i.e., controlling voltages and/or currents `arg1, arg2, ...`, and two expressions, `value_expression` and `derivative_expression`, to describe the value of an element and its derivative(s) (automatic evaluation of derivatives will be implemented later):

```
name  node+ node-  ARG(n) arg1 arg2 ... argn
+  {value_expression} {derivative_expression}
```

It should be noted that for capacitors and inductors the element "values" are the charge and flux, respectively; the capacitance of a capacitor and the inductance of an inductor correspond to the derivative of the charge with respect to the voltage over the capacitor (which must be the argument) or the derivative of the flux with respect to the current through the inductor. For multivariate elements, the "charge sharing" and "flux sharing" effects correspond to partial derivatives with respect to consecutive arguments (i.e., controlling voltages or currents).

The expressions can be more sophisticated than just arithmetic expressions as the class of operators includes the assignment ":=", the sequential composition ";", conditional

operators "**if**", "**then**", "**else**", all logical operators, standard functions (exp, ln, sin, etc.) and user-defined functions; the arguments (or controlling voltages and currents) `argi` in expressions are denoted by `#i, i=1,2,....`

The "exponential capacitance" used in the previous example can be described as a formula-driven capacitor in the following way:

```
Cexp 4 0  ARG(1) 4 0
+               { 1E-10*(#1+0.5*(exp(-#1)-1.0)) }
+               { 1E-10*(1.0-0.5*exp(-#1)) }
```

It should be observed that (usually) formula-driven elements are much easier to modify than the table-driven ones.

4. PROGRAM-DRIVEN SIMULATION

In this case the characteristics of (nonlinear) elements or blocks are specified by user-defined routines with controlling voltages and/or currents (and some other information) passed as parameters. The description is thus composed of two parts, the element's or block's "skeleton" that indicates the connections with the remaining part of the circuit, and a "programmed" user-supplied characteristic which defines the element by a corresponding section of code, identified in the "skeleton" description by the `FUN(idf)` parameter:

```
name node+ node- FUN(idf) ARG(n) arg1 arg2 ... argn
+  par1,par2,par3,...
```

`idf` is the evaluation function identifier, `ARG(n)` indicates the number of arguments `arg1, arg2, ...`, and `par1,par2,...` are additional parameters of evaluation routines.

The second part of specification is the evaluation routine itself which slightly changes from one class of elements to another. To accommodate these differences, there is one general evaluation routine for each class of elements, and the `idf` parameter is used for further identification within each class of functions. For (nonlinear) capacitive elements the evaluation routine is called `SPUNCE`, and it must be defined in a way equivalent to the following (FORTRAN) header:

```
 SUBROUTINE SPUNCE (IPS,IDF,VARG,NARG,VPAR,NPAR,VAL,
+                   MEV)
 DOUBLE PRECISION VARG(NARG),VPAR(NPAR(1))
 INTEGER NPAR(1)
```

where `IPS` is a unique internal identifier of the circuit element (i.e., `IPS` is a pointer to the element descriptor; there are routines which convert descriptor pointers into corresponding circuit element names and vice versa); `IDF` is the function identifier `idf` from the element description; `VARG` is the vector of controlling voltages and/or currents augmented by one more element (the first one) that passes the value of time for time-domain analysis, frequency for frequency-domain analysis, etc., so VARG(2)=value(arg1), VARG(3)=value(arg2), ...); `NARG` is the number of arguments `n+1`; `VPAR` is a vector of parameters `par1,par2,...` and `NPAR` is the number of parameters; `VAL` returns the value of required attribute, and `MEV` indicates the attribute to be evaluated; for `SPUNCE`, if MEV<0, VAL should return the value of charge, MEV=0 indicates that VAL should return the value of capacitance as required for time-domain analysis, if MEV>0, VAL should return the value of capacitance as required for small-signal frequency-domain analysis.

Program-driven evaluation of the "exponential capacitance" used in previous examples can be performed by the following routine (assuming `idf=9` is used for identification of this particular function):

```
      SUBROUTINE SPUNCE(IPS,IDF,VAR,NAR,VPA,MPA,
+                       VAL,MEV)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION VAR(1),VPA(1)
C ... IDF=3 denotes the exponential function
      IF (IDF.EQ.3) THEN
        VEXP=DEXP(-VAR(2))
        IF (MEV.LT.0) THEN
C ....... find the charge
          VAL=VPA(1)*(VAR(2)+VPA(2)*(VEXP-1.0))
        ELSE
C ....... find the capacitance
          VAL=VPA(1)*(1.0-VPA(2)*VEXP)
        ENDIF
      ELSE
C ..... other evaluation functions
      ENDIF
      RETURN
      END
```

5. A COMPARATIVE EXAMPLE

The following example compares the previous three approaches within one "testing" circuit composed of three similar RC sections with different implementations of exponential capacitors:

```
** example - exponential capacitors
VV 1 0 DC(1) AC(1) PULSE(1 0 2NS 5NS 2NS 10NS)
* ... table-driven capacitor
R2 1 2 100
C2 2 0 PWQ(1) 2 0 (0.00 0.000E+00, 0.01 4.520E-13,
+  0.03 1.010E-12, 0.05 2.039E-12, 0.07 3.088E-12,
+  0.10 5.242E-12, 0.30 1.704E-11, 0.50 3.033E-11,
+  0.70 4.483E-11, 1.00 6.839E-11, 2.00 1.568E-10,
+  3.00 2.525E-10, 5.00 4.503E-10)
* ... formula-driven capacitor
R3 1 3 100
C3 3 0 ARG(1) 3 0 { 1E-10*(#1+0.5*(exp(-#1)-1.0)) }
+                 { 1E-10*(1.0-0.5*exp(-#1)) }
* ... programmed capacitor
R4 1 4 100
C4 4 0 FUN(3) 1E-10 0.5
** analyses
.TR 2NS 20NS
.PRINT TR V(1) V(2) V(3) V(4)
.AC 1ME 10ME 100ME 1G
.PRINT AC V(2) V(3) V(4)
.END
```

```
***** AC ANALYSIS          TEMPERATURE : 27.00 DEG C

  FREQ       V(2)       V(3)       V(4)

1.00d+6   9.9867d-1  9.9869d-1  9.9869d-1
1.00d+7   8.8860d-1  8.8984d-1  8.8984d-1
1.00d+8   1.9019d-1  1.9142d-1  1.9142d-1
1.00d+9   1.9369d-2  1.9499d-2  1.9499d-2

***** TRANSIENT ANALYSIS  TEMPERATURE : 27.00 DEG C

  TIME       V(1)       V(2)       V(3)       V(4)

0.00d+0   1.0000d+0  1.0000d+0  1.0000d+0  1.0000d+0
2.00d-9   1.0000d+0  1.0000d+0  1.0000d+0  1.0000d+0
4.00d-9   6.0000d-1  9.5450d-1  9.5426d-1  9.5426d-1
6.00d-9   2.0000d-1  8.3011d-1  8.2973d-1  8.2973d-1
8.00d-9   0.0000d+0  6.4995d-1  6.4999d-1  6.4999d-1
1.00d-8   0.0000d+0  4.9327d-1  4.9335d-1  4.9335d-1
1.20d-8   0.0000d+0  3.6820d-1  3.6795d-1  3.6795d-1
1.40d-8   0.0000d+0  2.6845d-1  2.6955d-1  2.6955d-1
1.60d-8   0.0000d+0  1.8565d-1  1.9405d-1  1.9405d-1
1.80d-8   5.0000d-1  1.7182d-1  1.7854d-1  1.7854d-1
2.00d-8   1.0000d+0  3.7609d-1  3.7624d-1  3.7624d-1
```

Fig.2 shows the results of time-domain analysis; the plots for different implementations of the capacitor are practically identical.

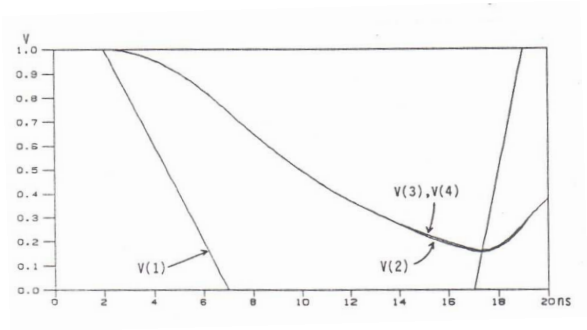

Fig.2. Results of time-domain analysis.

6. CONCLUDING REMARKS

The extensions described in this paper are not as general as – for example – the modelling language MAST used in the SABER simulation system [Getr], in which the so called "templates" define both the terminal relationships of generic components and the component contributions to the system of equations solved by the simulator. In SPICE-PAC the contributions of circuit elements are fixed by the class of elements (capacitors, controlled sources, etc.); the "flexibility" is thus restricted to terminal relationships which can be expressed using table-driven, formula-driven or program-driven methods. It should be noted that this restriction simplifies the specifications (examples of MAST specifications given in [ORou] do not look as friendly and "easy to use" as it is claimed [Getr]); actually, this restriction allows to introduce all the extensions within the original SPICE input language with very moderate syntax changes.

The table-driven approach is provided for experiment-oriented applications which use measurement data as well as predicted or estimated characteristics of some elements or blocks. The formula-driven approach is for theoretical studies in which flexibility of modifications of symbolic formulas is of primary interest. The program-driven approach is mainly for very "demanding" applications and for those users whose interests are in integration of software tools and exploring new problems.

The extensions presented in this paper are implemented in SPICE-PAC versions 2G6c.89 and beyond.

Acknowledgement

The Natural Sciences and Engineering Research Council of Canada partially supported this research through Operating Grant A8222, and Northern Telecom Canada through Memorial University Interaction Program.

R e f e r e n c e s

[Bart] D.L. Barton, "Behavioral descriptions in VHDL"; VLSI Systems Design, vol.9, no.6, pp.28-33, 1988.

[Davis] R. Davis, "Diagnostic reasoning based on structure and behavior"; Artificial Intelligence, vol.24, no.1-3, pp.347-410, 1984.

[DeLS] J.T. Deutsch, T.D. Lovett, M.L. Squires, "Parallel computing for VLSI circuit simulation"; VLSI Systems Design, vol.7, no.7, pp.46-52, 1986.

[DiNaV] S. Director, S.R. Nassif, L.M. Vidigal, "A new way to speed up circuit simulation"; Electronics, vol.59, pp.71-74, August 1986.

[FRS] U. Feldmann, K-G. Rauch, F. Steger, "Circuit simulation on vectorprocessors"; Proc. Int. Conf. on Computer Technology, Systems and Application, Hamburg, West Germany, pp.246-249, 1987.

[Getr] I. Getreu, "Analog modeling language spans all system design levels"; Electronic Design, vol.35, no.13, pp.95-104, 1987.

[Kuip] B. Kuipers, "Commonsense reasoning about causality: deriving behavior from structure"; Artificial Intelligence, vol.24, no.1-3, pp.169-203, 1984.

[Niess] C. Niessen, "Hierarchical design methodologies and tools for VLSI chips"; Proc. IEEE, vol.71, no.1, pp.66-75, 1983.

[ORou] R. O'Rourke, "Behavioral modeling of digital devices in an analog simulation environment"; VLSI Systems Design, vol.9, no.1, pp.16-25, 1988.

[Whar] D.J. Wharton, "Behavioral modeling in logic simulation"; VLSI Systems Design, vol.7, no.8, pp.46-54, 1986.

[WSSN] J. White, R. Saleh, A. Sangiovanni-Vincentelli, A.R. Newton, "Accelerating relaxation algorithms for circuit simulation using waveform Newton, iterative step size refinement, and parallel techniques"; Proc. of Int. Conf. on Computer-Aided Design, Santa Clara CA, pp.5-7, 1985.

[Zub] W.M. Zuberek, "SPICE-PAC version 2G6c - an overview"; Technical Report #8903, Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada A1C-5S7, 1989.