Technical Report #8903

SPICE–PAC version 2G6c
An Overview

by

W.M. Zuberek

Department of Computer Science
Memorial University of Newfoundland
St. John's, Canada A1B 3X5

March 1989

Department of Computer Science
Memorial University of Newfoundland
St. John's, Canada A1B 3X5
tel: (709) 737–8627
fax: (709) 737–2009

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

SPICE-PAC version 2G6c
An Overview

A b s t r a c t

SPICE-PAC is a simulation package that is upward compatible with the popular SPICE-2G circuit simulator; it accepts the same circuit description language (with only a few minor exceptions) and provides the same circuit analyses, but it also supports a number of extensions and refinements which are not available in the original SPICE program. The most important difference, however, between SPICE and SPICE-PAC is in their internal organizations; SPICE is a program with one, fixed sequence of operations while SPICE-PAC is a collection of loosely coupled simulation "primitives" that can be "composed" in many different ways, as required in a particular application.

This flexibility of SPICE-PAC is quite important in "integrated" applications, i.e., applications in which circuit simulation is combined with other software tools, for example, optimization methods, statistical analysis, symbolic simulation, high-level (e.g., behavioral) simulation, and so on.

This report briefly outlines main features of SPICE-PAC, compares SPICE-PAC with the SPICE circuit simulator, sketches SPICE-PAC's basic applications, and then presents several recent extensions to the package. Simple examples are used to illustrate these extensions.

A more detailed description of the package is given in a Technical Report: "SPICE-PAC 2G6c - User's Guide", available from the Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada A1C-5S7 (and also through anonymous `ftp` at `ftp.cs.mun.ca` in `/pub/techreports` as `tr-8902.ps.Z`).

1. Introduction

Computer-aided circuit analysis or circuit simulation [McCal,Peder,Vlach] has become an established and accepted tool in the area of circuit design, with the SPICE-2G program from the University of California at Berkeley [Cohen,Vlad] as one of the most popular and widely used "second-generation" circuit simulators. However, second generation circuit simulators are "batch oriented" programs, which means that even a minor change of the analyzed circuit requires a new, independent run of the simulator. Many applications, with interactive circuit analysis and circuit optimization as just two examples, require numerous analyses of a circuit with the same topology but with a number of variable circuit parameters. Values of these parameters may depend upon results of previous analyses, usually performed within the same simulation session (e.g., circuit optimization). Therefore a more flexible structure of the circuit simulator is needed, in which different analyses are performed "on demand", and in which there is a simple and efficient mechanism for accessing internal representation of circuit elements in order to modify their values. SPICE-PAC is a simulation package which is compatible with the SPICE simulator, and which satisfies these additional requirements.

2. SPICE and SPICE-PAC

SPICE-PAC is upward compatible with the SPICE-2G6 program. It means that SPICE-PAC accepts the same circuit description language as SPICE (with a few minor exceptions) and provides the same set of circuit analyses, but it also supports a number of new features, not available in the original SPICE simulators. Examples of these extensions include:

- static and dynamic circuit variables; circuit variables are those attributes of circuit elements that can be modified during a simulation session; static circuit variables must be defined within the circuit description, and these definitions are used by the package to implement a very efficient access to such variables, as required in circuit optimization; dynamic circuit variables do not require any definition, so they are very flexible (as required in interactive applications) but relatively slow [Zvar],
- dynamic (i.e., at the "simulation-time") definitions of analyses, their parameters and outputs,
- a uniform hierarchical naming scheme in which levels of subcircuits are indicated by "qualifiers", i.e., `X1.X2.R123` denotes the element `R123` in the subcircuit `X2` of the subcircuit `X1`; subcircuit elements can be used in parameter lists and output specifications,
- parameterized subcircuit invocations; subcircuit definitions can be modified by parameters passed to the subcircuit expansion phase [Zpar],
- an interface to libraries of standard modules; standard modules are in the form of subcircuits stored in independent files within a file system, and they are accessed by (parameterized) module invocations [Zlib],

- enhanced circuit elements, i.e., circuit elements with characteristics defined by users in the form of formulas or tables of values, etc. [Zsour],
- enhanced analyses, i.e., circuit analyses which are extended by user-defined operations [Ztdom].

However, the most important difference between the SPICE program and the SPICE-PAC package is in their internal organizations. SPICE is a program with one, fixed sequence of analyses, indicated by appropriate parameters within the circuit description. The order of these analyses as well as representation of results are always the same since they are "built-into" the simulation program [Cohen]. SPICE-PAC, on the other hand, is a rather loosely coupled collection of "simulation primitives" (implemented by different components of the package), which can be combined together in many different ways (similarly to building a variety of structures from a set of LEGO-type elements). Typical examples of such "simulation primitives" include reading a circuit description, performing one of circuit analysis, changing values of (some) circuit elements, or redefining analysis parameters. The operations of the package are thus performed "on demand", as required by a particular application. In the case of interactive simulation, it is the user who - during a simulation session - selects the order, type and parameters of analyses. This means that this type of circuit simulation provides the user with a "feedback" which is unavailable in traditional "batch-oriented" simulators; within one interactive simulation session, the results of one analysis can be used to determine the new values of (some) circuit elements as well as the subsequent analyses and their parameters. In cases of "integrated" applications, when the simulation package is used as a "generator" of circuit responses (e.g., circuit optimization in which it evaluates the objective function and - possibly - the constraint functions [Poiv,Zopt]), the sequence of package operations, their types and parameters are determined by other software tools, "integrated" with the simulation package.

This flexible structure of the package makes it possible to combine the same set of "standard" analyses with several input processors accepting different forms of circuit specification (e.g., the SPICE input language, a functional-type circuit description, an output of a circuit extractor, etc.), to represent the results in different ways (graphical for the user, binary for further processing by other tools, textual for storing in a file, and so on). Furthermore, it is possible to replace some of the "standard" modules by dedicated user-defined methods, specialized to particular applications.

3. General organization

SPICE-PAC is organized in two major levels of subroutines, the so called "main" (or interfacing) subroutines which constitute the "simulation interface", and a collection of internal subroutines and functions. The main subroutines are called SPICEA, SPICEB, ..., SPICEY, and they perform "simulation primitives", such as reading and processing circuit descriptions (SPICEA), definitions of circuit variables (SPICEB), circuit analysis (SPICER or SPICES), etc.; the list of main subroutines and their functions is given in the Appendix

A. Each main subroutine invokes a number of internal subroutine and functions, which, however, are "invisible" to users.

SPICE-PAC is a package in which the "communication" with other packages and/or programs uses "internal" (or binary) representation of information. This means that parameters passed to the package as well as results returned from the package are stored in variables and arrays defined in an external "driving" program, and it is this external program that must perform all required conversions and all input/output operations (with the exception of circuit descriptions which are processed by the package). Consequently, there are no "printing" or "plotting" facilities built into the package, and any required form of "output" has to be provided by the external "driving" routines.

4. Input language

SPICE-PAC accepts circuit descriptions in the SPICE-2G form with the following few exceptions:

- `.TEMPERATURE` lines, for SPICE-PAC, can define one temperature only (the SPICEM subroutine should be used for subsequent definitions of the temperature),
- `.DC` lines, for SPICE-PAC, can describe one set of the DC transfer curve source and sweep limits (the SPICED subroutine should be used for subsequent definitions of DC analysis parameters),
- `.ALTER` sections and
- `.PLOT` lines, in SPICE-PAC, are simply ignored.

It should be noted that all control lines in circuit descriptions (`.DC` ..., `.AC` ..., etc.) are used only to define parameters of the corresponding analyses, while the analyses are performed (selectively) by calling main subroutines (SPICER or SPICES) with appropriate arguments. Moreover, all parameters defined by control lines in the circuit description can be redefined by appropriate subroutines of the package (SPICED, SPICEF, etc.), or can be replaced by parameters "predefined" in an extended circuit description, and activated when required (by the SPICEL subroutine).

Several modifications and extensions of the circuit description language (or the input language) are described in subsequent sections.

5. Extended circuit description

Extended circuit description is an optional part of the input file which follows the "basic" circuit description, and which can contain:

- definitions of (static) circuit variables,

- definitions of parameters and outputs for different analyses,
- declarations of pointers to circuit elements, as required by some SPICE-PAC subroutines (SPICER, SPICES), definitions of constants, requests of monitoring, exception handling directives, etc.

All information provided by an extended circuit description can also be obtained by appropriate calls of SPICE-PAC subroutines. In most cases, however, extended circuit descriptions can significantly simplify the use of SPICE-PAC; the use of extended circuit description allows the "main" program to be more general since all specific, circuit-dependent information can be placed in the data file rather than incorporated directly into the program body.

Extended circuit description is separated from the (basic) circuit description by:

```
.END/EXT
```

and is terminated by:

```
.END
```

Circuit variables are defined by the "VAR" lines:

```
.VAR  variable-name
```

where `variable-name` is either a simple element name for those elements which have one attribute only (usually it is the "value" of the circuit element, e.g., the resistance of a resistor or the capacitance of a linear capacitor), or a composite name which is used for multi-attribute circuit elements to indicate:

- polynomial coefficients of nonlinear capacitors and inductors (e.g., `C15'#3`),
- polynomial coefficients of dependent voltage and current sources (e.g., `E1'#0`),
- DC and AC parameters of independent voltage and current sources (e.g., `VIN'DC`),
- parameters of time-dependent functions of voltage and current sources (e.g., if an independent voltage source is described as "`VIN 3 0 PULSE(0,1,0,1NS,1NS,5NS)`" then `VIN'#4` denotes the "fall time" of `VIN`),
- parameters of semiconductor devices (e.g., `M1'L`),
- parameters of (common) device models (e.g., `MOD'RS`),
- parameters of models associated with (specific) semiconductor devices (e.g., `M2:RS`); in this case "common" model parameters are not influenced by changes of "specific" device model parameters.

Simple and composite variable names can be direct or qualified. Direct names are used for those elements which are at the "top" (or "main") level of circuit description (i.e., elements not belonging to subcircuits). Elements of subcircuits must be identified by qualified names in which the element name follows the full sequence of the subcircuit names separated by periods "." (starting from the "top" level); for example, `X1.X3.X2.M12:VTO` is a composite qualified variable name denoting the threshold voltage at zero bias (`VTO`) of the MOS transistor `M12` in the subcircuit `X2` of the subcircuit `X3` of the subcircuit `X1`.

Parameters for SPICE-PAC analyses and their outputs are defined using the following form:

```
.PAR/id  analysis(parameters)
.OUT/id  analysis(output-list)
```

where `id` is an unsigned integer number that is used as an identifier of the definition, `analysis` is `DC`, `TR`, `AC`, `NO`, `DI`, `FO`, `TF`, `DS` or `AS` for DC transfer curve, transient, small-signal AC, noise, distortion, Fourier, DC transfer function, DC and AC sensitivity analyses, respectively, and `parameters` is a list of corresponding parameters separated by commas:

```
DC(source-name,Vistrt,Vistop,Nrstep)
TR(Tmstrt,Tmstop,Nrstep,Stpmax,Incond)
AC(freq1,freq2,....,freqn)
NO(source-name,output-var,Incrmt)
DI(rload-name,Fratio,Fampl,Refpwr,Incrmt)
FO(Frqval,Nrharm,Tfstrt,Nrstep,Stpmax,Incond)
TF(source-name,Mkrout)
DS(R-name1,...,Q-name1,....)
AS(Sens-out,SR-name1,...,SI-name1,...)
```

where `source-name` is a simple (direct or qualified) name of an independent voltage or current source, `output-var` is an output variable which defines the summing point for equivalent output noise, `rload-name` is a simple (direct or qualified) name of an output load resistor for distortion analysis, `freq1`,`freq2`, ..., are either simple frequency values or linear or logarithmic subranges in the form:

```
LIN(Nrstep,Frstrt,Frstop)
LOG(Nrstep,Frstrt,Frstop)
```

and `Nrstep` is the required number of steps in a subrange (including boundaries), `Frstrt` is the initial frequency and `Frstop` is the final frequency of a subrange. Some examples:

```
.PAR/11 DC(VIN,-5.0,5.0,21)
.PAR/12 DC(VIN,-0.1,0.1,21)
.OUT/19 DC(V(1),V(5,2))
.PAR/21 AC(LOG(11,1.D2,1.D4),1.D5,LOG(11,1.D6,1.D8))
.OUT/29 AC(VM(5),VP(5),VDB(3,2))
.PAR/53 NO(VIN,V(X2.4),2)
```

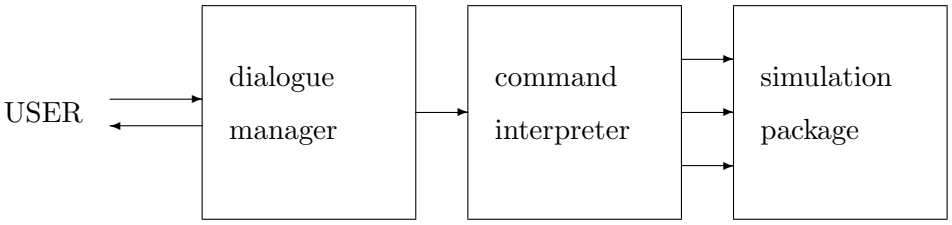
6. Examples of applications

Interactive simulation and circuit optimization were two basic goals of the original SPICE-PAC project. Numerous further refinements and extensions to the package opened several new domains of interesting applications. Mixed-mode simulation, or simulation of mixed, analog-digital circuits is the most recent one.

6.1. Interactive simulation

In interactive simulation, the analyses are performed "on demand", as indicated by the user who also selects the parameters and outputs. Usually, this selection is based on results obtained from previous analyses, most likely within the same simulation session.

The organization of an interactive simulator can be outlined as a three-layer structure composed of a "dialogue manager", "command interpreter" and the simulation package:



"Dialogue manager" mainly organizes the interaction with the user, selects prompts (or menus), checks formal correctness of entered commands, and invokes command interpreter(s) to perform the required operations. "Command interpreter" analyzes user-supplied commands and translates them into equivalent sequences of simulation primitives; it also returns the results either as generated by the simulation package, or obtained from a "post-processor".

The following example shows simple elements of interactive simulation in the "standard" interactive driver, distributed with the SPICE-PAC package (RE and RB are used as "dynamic circuit variables"):

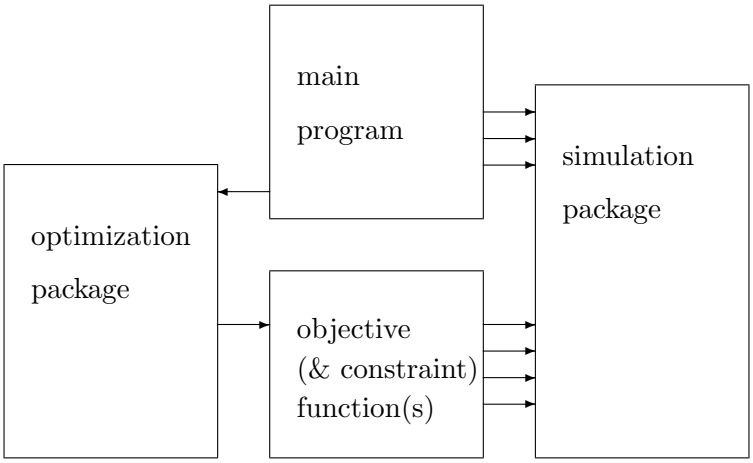
```
****     SPICE-PAC.2G6c:89.02a (MUN:Xg)   DATE : 21 FEB 89 AT 16:13:58
****     INPUT LISTING                    TEMPERATURE =   27.000 DEG C
** single stage CE amplifier
VCC 5 0 12
VIN 1 0 AC 1
RB 2 5 750K
RE 3 0 150
RC 4 5 5K
CC 4 5 100PF
CB 1 2 10UF
Q1 4 2 3 MOD
.MODEL MOD NPN(BF=100 VAF=50 IS=1.E-9 RB=100)
.PRINT AC V(4) VP(4) VDB(4)
```

```
.AC 100,50K,10MEG
.END

enter operation (or /) : .ac

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

      FREQ        V(4)     V.P(4)    V.DB(4)
    1.00d+02    2.92d+01 -1.79d+02  2.93d+01
    5.00d+04    2.89d+01  1.71d+02  2.92d+01
    1.00d+07    9.45d-01  9.19d+01 -4.96d-01

enter operation (or /) : .var(RE)=100
enter operation (or /) : .ac

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

      FREQ        V(4)     V.P(4)    V.DB(4)
    1.00d+02    4.15d+01 -1.79d+02  3.24d+01
    5.00d+04    4.10d+01  1.71d+02  3.23d+01
    1.00d+07    1.35d+00  9.19d+01  2.61d+00

enter operation (or /) : .var(RB)=1D6
enter operation (or /) : .var(RE)=75
enter operation (or /) : .ac

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

      FREQ        V(4)     V.P(4)    V.DB(4)
    1.00d+02    5.03d+01 -1.79d+02  3.40d+01
    5.00d+04    4.97d+01  1.71d+02  3.39d+01
    1.00d+07    1.64d+00  9.19d+01  4.30d+00

enter operation (or /) : .op

***** OP-POINT INFORMATION           TEMPERATURE :  27.00 DEG C

    0.00d+00  V(1)
    4.59d-01  V(2)
    9.64d-02  V(3)
    5.63d+00  V(4)
    1.20d+01  V(5)
   -1.29d-03  VCC
    0.00d+00  VIN
```

```
enter operation (or /) : and so on ...
undefined operation.
```

6.2. Circuit optimization

In the case of circuit optimization, there are (at least) two packages that need to be coordinated, the optimization package and the simulation package, the latter used to evaluate the objective (and possibly constraint) functions at points determined by the optimization algorithm. If "indirect communication" is used, the optimization package "supervises" the whole optimization process and it invokes the evaluation routines whenever the values of the objective function, constraints or their derivatives are needed:



In the following optimization example, it is to find the values of biasing resistors RB and RE in a single-stage CE amplifier such that for the midband frequency f=50KHz the voltage-gain is maximum, the input resistance is not less than 10Kohms, and the output voltage swing is at least 10V peak-to-peak.

Since the voltage-gain is to be maximized, the (single) objective function is equal to the negative value of the voltage-gain provided by the AC analysis [Zopt]. There are 3 inequality (nonlinear) constraints; one describes the required input resistance, also determined by the AC analysis, while the remaining two constraints deal with the "positive" and "negative" part of the required output voltage swing determined by the node voltages obtained from the DC operating point analysis. The extended circuit description is used to indicate the optimization variables (it could also "pass" other circuit parameters used in the optimization process):

```
****     SPICE-PAC.2G6c:89.02a (MUN:Xm)   DATE : 14 FEB 89 AT 12:33:51
****     INPUT LISTING                    TEMPERATURE =   27.000 DEG C
** single stage CE amplifier - AC/DC optimization
VCC 5 0 12
```

```
 VIN 1 0 AC 1
 RB 2 5 750K
 RE 3 0 150
 RC 4 5 5K
 CB 1 2 100UF
 Q1 4 2 3 MOD
 .MODEL MOD NPN(BF=100 VAF=50 IS=1.E-9 RB=100)
 .PRINT AC V(4) V(2) I(VIN)
 .AC 50K
 .END/EXT
 .VAR RB
 .VAR RE
 .END
```

```
nr     RB       RE    -obj.fun  c.fun.1  c.fun.2  c.fun.3
 0  7.50d+05 1.50d+02 2.92d+01-1.71d+00 3.09d+00 5.44d+03
 1  7.50d+05 9.38d+01 4.38d+01-1.68d+00 3.16d+00 3.89d+02
 2  7.50d+05 8.95d+01 4.55d+01-1.68d+00 3.16d+00-2.60d+00
 3  7.50d+05 8.98d+01 4.54d+01-1.68d+00 3.16d+00 2.25d+01
 4  7.52d+05 8.95d+01 4.55d+01-1.67d+00 3.15d+00-1.36d-02
 5  7.58d+05 8.91d+01 4.56d+01-1.61d+00 3.09d+00-2.51d-01
 6  7.90d+05 8.73d+01 4.62d+01-1.31d+00 2.81d+00-6.30d+00
 7  9.36d+05 7.94d+01 4.86d+01-2.19d-01 1.75d+00-1.01d+02
 8  9.65d+05 7.91d+01 4.85d+01-3.50d-02 1.57d+00-9.20d+00
 9  9.71d+05 7.90d+01 4.85d+01-1.10d-03 1.53d+00-8.85d-01
10  9.71d+05 7.90d+01 4.85d+01-5.74d-06 1.53d+00-1.93d-03
.....................................................
22  1.31d+06 6.73d+01 5.15d+01 1.58d+00 1.31d-05-4.84d-03
```

The (partial) trace of the optimization shows that the results after 10 iteration steps are quite good and that the subsequent 12 steps improve the solution rather insignificantly. After 22 iteration steps, only the third constraint (input resistance) is "violated" by less than 5 milliohms, and the voltage gain has been improved from initial 29.2V/V to 51.3V/V.

The evaluation routine is as follows:

```
      SUBROUTINE PROC (N,L,X,F,Y,K)
C   N - the number of optimization/circuit variables,
C   L - the number of (nonlinear) constraints,
C   X - the vector of optimization/circuit variables,
C   F - the value of the objective function,
C   Y - the vector of constraint values,
C   K - communication flag ("stage" of the optimization method).
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION X(N),Y(L),YTAB(5)
      COMMON /VARIDS/ NROUTF,NR,ITAB,KTAB(5)
```

```
C ... update variables
      CALL SPICEU(KTAB,X,N,IEX)
      IF(IEX.NE.N) CALL ERROR('U',IEX,*90)
C ... perform DC-OP analysis
      CALL SPICER(0,YTAB,XTAB,5,0,IR,IC,IEX)
      IF(IEX.LT.0) CALL ERROR('R/OP',IEX,*90)
      Y(1)=DABS(YTAB(4)-YTAB(2))-5.0
      Y(2)=DABS(YTAB(5)-YTAB(4))-5.0
C ... perform AC analysis
      CALL SPICER(3,XTAB,YTAB,1,3,IR,IC,IEX)
      IF(IEX.NE.0) CALL ERROR('R/AC',IEX,*90)
      F=-YTAB(1)
      Y(3)=YTAB(2)/YTAB(3)-1.D4
C ... print the trace
      NR=NR+1
      WRITE(NROUTF,100) NR,(X(I),I=1,N),-F,(Y(I),I=1,L)
  100 FORMAT(1X,I1,1X,1P7D9.2)
      RETURN
   90 STOP
      END
```

7. Enhanced circuit elements

Enhanced circuit elements are elements with "nonstandard" characteristics, described by user-supplied data tables (for table-driven elements) or formulas (for analytic characteristics). Two types of SPICE-PAC's enhanced elements, dependent voltage and current sources, and nonlinear capacitors and inductors, are described in greater detail. A new "TABLE" pseudoelement is also presented; it has been implemented to allow sharing the data by several table-driven elements.

7.1. TABLE pseudoelements

Quite often several "table-driven" circuit elements can use the same data (similarly to several semiconductor devices "sharing" the same model). In such cases it is convenient (and recommended) to define the "shared" data as an independent (multidimensional) array described by a "TABLE" pseudoelement, indicated in element descriptions by the "USE" specifier (see the next section).

The general syntax of "TABLE" descriptions is

```
.TABLE  tname  ARG(narg)  DIM(d1,d2,...)  p0,p1,p2,...
```

in which

`tname` is the (unique) name of this pseudoelement,

`ARG(narg)` specifies the number of arguments (or "independent variables) `narg` (with the default value equal to 1); this parameter is mandatory if the number of arguments is greater than 1,

`DIM(d1,d2,...)` is an optional specification of the "organization" of the following elements `p0,p1,p2,...`; if the elements are organized into a multidimensional array, the dimensions of this array can be indicated in the `DIM` option; when this option is used, it must contain `narg` (positive) integer values `d1,d2,...,dnarg` which specify the consecutive dimensions of the data array, and then the number of elements `p0,p1,p2,...` must be equal to (d1+d2+...+dnarg+d1*d2*...*dnarg), i.e., `narg` vectors of independent variable values (or vectors of arguments values), followed by the corresponding array of (dependent) data; the ordering of data must be consistent with their uses since SPICE-PAC simply enters consecutive elements into consecutive positions of an internal vector, and then passes this vector to the (user-defined) function evaluation routine,

`p0,p1,p2,...` are consecutive numerical data elements.

The following simple example approximates a voltage dependent capacitance by a sequence of 12 points which are pairs (controlling voltage, capacitance), with increasing values of controlling voltages (parentheses may be used rather arbitrarily for "grouping" the data since they are ignored by the SPICE-PAC's input processor):

```
.TABLE  tabvcap1  ARG(1)       (0.0 5.00d-11, 0.1 5.48d-11,
+   0.2 5.91d-11, 0.3 6.30d-11, 0.4 6.65d-11, 0.5 6.97d-11,
+   0.6 7.26d-11, 0.8 7.75d-11, 1.0 8.16d-11, 1.2 8.49d-11,
+   1.5 8.88d-11, 2.0 9.32d-11, 3.0 9.75d-11, 5.0 9.97d-11)
```

If the values of controlling voltages and corresponding capacitances are to be used as separate vectors (say, first all controlling voltages, and then the corresponding capacitances), the `DIM` option should be specified with one argument (controlling voltage), and then the same coefficients should be arranged in a different way:

```
.TABLE  tabvcap2  ARG(1)  DIM(12)
+  (0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.8 1.0 1.2 1.5 2.0 3.0 5.0)
+  (5.00d-11 5.48d-11 5.91d-11 6.30d-11 6.65d-11 6.97d-11
+   7.26d-11 7.75d-11 8.16d-11 8.49d-11 8.88d-11 9.32d-11
+   9.75d-11 9.97d-11)
```

7.2. Enhanced dependent sources

The SPICE program defines four types of dependent (nonlinear) sources (or transducers) [Vlad] characterized by the following equations

$$\mathbf{v=e(v),\ \ i=f(i),\ \ \ i=g(v),\ \ \ v=h(i)}$$

where the functions **e**, **f**, **g** and **h** must be polynomials, and the arguments **v** and **i** may be multidimensional; the polynomial functions are specified by the sets of their coefficients. The general form of SPICE dependent source description is

```
Zname  node+ node-  POLY(narg) arg1,arg2,...  p0,p1,p2,...  IC=...
```

where `Zname` is the name of a source and `Z` is `E`, `F`, `G` or `H` for different types of controlled sources, `node+` and `node-` are the positive and negative nodes of the source, `POLY(narg)` must be used for sources with more than one argument (otherwise it is optional), and `narg` is the number of arguments; each argument `arg1`, `arg2` ..., is either an independent voltage source name "Vname" (for current controlled sources) or a pair of node numbers "vnode+ vnode-" (for voltage controlled sources), `p0,p1,...,` are consecutive polynomial coefficients, and the last part is an optional specification of initial conditions `IC=...`.

Enhanced dependent sources can be described in one of the two following forms:

```
(1)   Zname  node+ node-  FUN(idf)  ARG(narg) arg1,arg2,...
      +                   DIM(d1,d2,...)  p0,p1,p2,...  IC=...

(2)   Zname  node+ node-  FUN(idf)  ARG(narg) arg1,arg2,...
      +                   USE(tname)  IC=...
```

where:

`FUN(idf)` defines a numerical identifier `idf` which is passed to the function evaluation routine (and its default value is 1); usually `idf` denotes the "method" of evaluation, e.g., "idf=1" may indicate unidimensional linear interpolation, "idf=2" unidimensional quadratic interpolation, "idf=5" two-dimensional linear interpolation, etc.; the interpretation and use of `idf` depends only on the evaluating routines;

`ARG(narg)` specifies the number of arguments `narg` (with the default value equal to 1); this parameter is mandatory if the number of arguments is greater than 1;

`DIM(d1,d2,...)` is as for "TABLE" pseudoelements,

`p0,p1,...` are consecutive data elements,

`USE(tname)` is a reference to a "TABLE" of shared coefficients which is identified by the name `tname`.

The list of coefficients `p0,p1,...` may thus be used to specify arbitrary parameters which are passed to the evaluation routine together with some "identification" information.

Both `FUN(idf)` and `ARG(narg)` are optional parts but at least one of them must be present in the description of enhanced dependent sources.

User-supplied enhancing functions must be defined by a SPUDSE routine in a way equivalent to the following (Fortran) header:

```
SUBROUTINE SPUDSE (IPS,IDF,VARG,NARG,VPAR,NPAR,VAL,PDER)
DOUBLE PRECISION VARG(NARG),VPAR(1),VAL,PDER
INTEGER IPS,IDF,NARG,NPAR(1)
```

where `IPS` is a unique internal identifier of the voltage/current source (i.e., `IPS` is a pointer to the dependent source descriptor; SPICE-PAC provides subroutines which convert descriptor pointers into corresponding external element names - SPICEY - and external element names into corresponding descriptors - SPICEP); `IDF` is the function identifier `idf` from the source description; `VARG` is a vector which on entry is set to indicated controlling voltages or currents, VARG(1)=value(arg1), VARG(2)=value(arg2), ...), and which returns the values of partial derivatives of the controlled voltage or current with respect to consecutive arguments; `NARG` is the number of arguments, `narg`; `VPAR` is a vector of coefficients specified in the source description (i.e., VPAR(1)=p0, VPAR(2)=p1, ...); `NPAR` is a vector which contains at least 2 elements; the first element, NPAR(1), is the number of coefficients in `VPAR`; if the second element, NPAR(2), is equal to zero, the NPAR vector has no continuation, otherwise the elements NPAR(2),...,NPAR(NARG+1) contain consecutive dimensions indicated in the `DIM` option, i.e., NPAR(2)=d1, NPAR(3)=d2, etc.; `VAL` is a variable which returns the value of the controlled voltage or current, and `PDER` is a variable which returns the (scalar) product of partial derivatives and arguments.

The following example shows a unidimensional table-driven voltage source that describes the transfer characteristic of a MOSFET inverter:

```
****     SPICE-PAC.2G6c:89.03  (MUN:Xg)   DATE : 27 MAR 89 AT 18:13:25
****     INPUT LISTING                    TEMPERATURE =   27.000 DEG C
* The design and analysis of VLSI circuits (Glasser,Dobberpuhl), p.139.
* DC transfer curve analysis - controlled volage sources
VIN 1 0 0V
VDD 3 0 5V
.OPTIONS DEFL=2.25E-6
* MOSFET inverter - reference output
M1 2 1 0 0 NENHS W=11.2U AD=61P PD=42U
M2 3 2 2 0 NDEPS W=4.2U L=6.25U
.MODEL NENHS NMOS LEVEL=3 RSH=0 TOX=330E-10 LD=0.19E-6 XJ=0.27E-6
+      VMAX=13E4 ETA=0.25 KAPPA=0.5 NSUB=5E14 UO=650 THETA=0.1
+      VTO=0.946 CGSO=2.43E-10 CGDO=2.43E-10 CJ=6.9E-5 CJSW=3.3E-10
+      PB=0.7 MJ=0.5 MJSW=0.3 NFS=1E10
.MODEL NDEPS NMOS LEVEL=3 RSH=0 TOX=330E-10 LD=0.19E-6 XJ=0.27E-6
+      VMAX=13E4 ETA=0.25 KAPPA=0.5 NSUB=50E14 UO=650 THETA=0.04
+      VTO=-2.078 CGSO=2.43E-10 CGDO=2.43E-10 CJ=6.9E-5 CJSW=3.3E-10
+      PB=0.7 MJ=0.5 MJSW=0.3 NFS=1E10
* source-1 : E1 - table-driven source with linear interpolation
E1 4 0 FUN(1) 1 0 USE(TMOSPAIR)
R1 4 7 1K
V1 7 0 0V
```

```
 * source-2 : E2 - table-driven source with quadratic interpolation
 E2 5 0 FUN(2) 1 0 USE(TMOSPAIR)
 R2 5 8 1K
 V2 8 0 0V
 * MOSFET data:
 .TABLE TMOSPAIR
 +  (0.0 5.00d+00,0.5 5.00E+00,0.8 5.00E+00,0.9 4.96E+00,
 +   1.0 4.86E+00,1.1 4.71E+00,1.2 4.47E+00,1.3 4.10E+00,
 +   1.4 3.13E+00,1.5 1.66E+00,1.6 5.07E-01,1.7 3.60E-01,
 +   1.8 2.96E-01,1.9 2.56E-01,2.0 2.27E-01,2.1 2.05E-01,
 +   2.2 1.87E-01,2.3 1.73E-01,2.4 1.61E-01,2.5 1.51E-01,
 +   2.7 1.34E-01,3.0 1.17E-01,3.5 9.66E-02,4.0 8.36E-02,
 +   4.5 7.43E-02,5.0 6.74E-02)
 * DC analysis
 .DC VIN 0.25 4.75 0.5
 .PRINT DC V(2),V(2,4),V(2,5)
 .END

 ***** DC TRANSFER CURVE             TEMPERATURE :  27.00 DEG C

        VIN        V(2)     V(2,4)    V(2,5)

     2.50d-01    5.00d+00 -1.96d-07 -1.96d-07
     7.50d-01    5.00d+00 -3.20d-04 -1.28d-02
     1.25d+00    4.31d+00  2.61d-02 -4.89d-02
     1.75d+00    3.24d-01 -4.30d-03 -1.30d-03
     2.25d+00    1.80d-01 -3.55d-04 -1.05d-04
     2.75d+00    1.31d-01 -1.48d-04  1.00d-04
     3.25d+00    1.05d-01 -1.36d-03 -4.37d-04
     3.75d+00    8.95d-02 -5.97d-04 -1.34d-04
     4.25d+00    7.86d-02 -3.66d-04 -6.63d-05
     4.75d+00    7.06d-02 -2.29d-04  7.07d-05
```

The implementation of interpolation used by table-driven sources (preceded by a binary search, if needed), can be as follows:

```
      SUBROUTINE SPUDSE (IPS,IDF,VARG,NARG,VPAR,NPAR,VAL,PDER)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION VARG(NARG),VPAR(1),NPAR(1)
      DATA IND / 1 /
      ARG=VARG(1)
      N=NPAR(1)
      IF (IDF.EQ.1) THEN
C ... idf=1 => table-driven element with linear interpolation
```

```
        IF (NARG.NE.1) GO TO 90
C ..... first check if the new point is close to the last one
        IF (ARG.GE.VPAR(IND)) THEN
          IF (ARG.LE.VPAR(IND+2)) GO TO 20
          IF (IND.EQ.N-3) GO TO 20
          IF (IND.LT.N-3) THEN
            IND=IND+2
            IF (ARG.LE.VPAR(IND+2)) GO TO 20
          ENDIF
        ENDIF
C ..... binary search
        I=1
        J=N/2
   10   K=(I+J)/2
        IND=K+K-1
        IF (I.NE.K) THEN
          IF (ARG.LT.VPAR(IND)) THEN
            J=K
          ELSE
            I=K
          ENDIF
          GO TO 10
        ENDIF
C ..... interpolation
   20   DEL=VPAR(IND+2)-VPAR(IND)
        VAL=VPAR(IND+1)
        DER=0.D0
        IF (DEL.GT.0.D0) THEN
          DER=(VPAR(IND+3)-VAL)/DEL
          VAL=VAL+DER*(ARG-VPAR(IND))
        ENDIF
        VARG(1)=DER
        PDER=ARG*DER
      ELSE
C ... enhancing functions for other idf values
      ENDIF
      RETURN
   90 WRITE(IOFILE,900) IPS,IDF
  900 FORMAT(' ... SPUDSE : incorrect arguments for :',I4,' fun:',I3)
      STOP
      END
```

7.3. Enhanced capacitors and inductors

The only function that is supported by SPICE for nonlinear capacitors and inductors is a polynomial (with capacitor's voltage or inductor's current as its only variable). The general form of SPICE description is

```
Zname   node+ node-   POLY  p0,p1,p2,...   IC=...
```

where `Z` is `C` for capacitors and `L` for inductors, `POLY` indicates a nonlinear element, polynomial coefficients are given as `p0,p1,p2,...,`, and `IC=...` is an optional definition of the initial condition.

Enhanced capacitors and inductors can be described using one of the following two variants:

```
(1)   Zname  node+ node-  FUN(idf)  ARG(narg) arg1,arg2,...
      +                   DIM(d1,d2,...)  p0,p1,p2,...  IC=...

(2)   Zname  node+ node-  FUN(idf)  ARG(narg) arg1,arg2,...
      +                   USE(tname)  IC=...
```

where all parameters are as before.

User-supplied enhancing functions for capacitors and inductors are specified within subroutines SPUNCE and SPUNLE, respectively. The subroutines must conform to the following (Fortran) header:

```
SUBROUTINE SPUNxE (IPS,IDF,VARG,NARG,VPAR,NPAR,VAL,MOPR)
DOUBLE PRECISION VARG(NARG),VPAR(1),VAL
INTEGER IPS,IDF,NARG,NPAR(1),MOPR
```

where `x` is `C` or `L` for capacitors and inductors, respectively; `IPS`, `IDF`, `NARG`, `VPAR`, `NPAR` and `VAL` are as for SPUDSE; `VARG`, the vector of arguments, always contains at least two elements since VARG(1) passes the current value of (simulated) time for transient analysis or radial frequency for (small signal) AC analysis, VARG(2) is the value of the element's voltage (for capacitors) or current (for inductors), while the consecutive elements (if any) are the values of indicated controlling voltages and currents, VARG(3)=value(arg1), VARG(4)=value(arg2), ...; finally, `MOPR` indicates the required operation; if MOPR=0, VAL should return the capacitance (or inductance) which is required for transient analysis, if MOPR<0, VAL should return the charge (or flux) of the element if MOPR>0, VAL should return the capacitance (or inductance), as required for small signal AC analysis.

The following simple example compares two implementations, analytical and table-driven, of a nonlinear capacitor in which capacitance is an exponential function of the capacitor voltage Vcap, C=p0*(1-p1*exp(-p2*Vcap)), where "p0" "p1" and "p2" are parameters (100pF, 0.5 and 1.0, respectively):

```
****  SPICE-PAC.2G6c:88.05 (MUN:X)     DATE : 11 MAY 88 AT 18:06:47
****  INPUT LISTING                    TEMPERATURE =   27.000 DEG C
VV 1 0 DC=5 PULSE(0 5 2NS 2NS 2NS 10NS 20NS)
* ... linear capacitor (for comparison)
R1 1 2 100
C1 2 0 100PF
* ... exponential (analytical) capacitance, idf=9:
R2 1 3 100
C2 3 0 FUN(9) 1E-10 0.5 1.0
* ... table-driven capacitance; idf=2 - quadratic interpolation
R3 1 4 100
C3 4 0 FUN(2) ARG(1) 4 0 USE(TCexp)
* table entries are triples <voltage, capacitance, charge>
.TABLE TCexp  ARG(1)          (0.00  5.000E-11  0.000E+00,
+  0.01  5.050E-11  4.520E-13, 0.03  5.148E-11  1.010E-12,
+  0.05  5.244E-11  2.039E-12, 0.07  5.338E-11  3.088E-12,
+  0.10  5.476E-11  5.242E-12, 0.30  6.296E-11  1.704E-11,
+  0.50  6.967E-11  3.033E-11, 0.70  7.517E-11  4.483E-11,
+  1.00  8.161E-11  6.839E-11, 2.00  9.323E-11  1.568E-10,
+  3.00  9.751E-11  2.525E-10, 5.00  9.966E-11  4.503E-10)
* ... transient analysis
.TR 2NS 20NS 0
.PRINT TR V(1) V(2) V(3) V(4)
.END

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

      TIME        V(1)      V(2)      V(3)      V(4)
    0.00d+00    0.00d+00  0.00d+00  0.00d+00  0.00d+00
    2.00d-09    0.00d+00  0.00d+00  0.00d+00  0.00d+00
    4.00d-09    5.00d+00  4.68d-01  7.47d-01  7.52d-01
    6.00d-09    5.00d+00  1.29d+00  1.66d+00  1.68d+00
    8.00d-09    5.00d+00  1.96d+00  2.31d+00  2.33d+00
    1.00d-08    5.00d+00  2.51d+00  2.82d+00  2.83d+00
    1.20d-08    5.00d+00  2.96d+00  3.23d+00  3.23d+00
    1.40d-08    5.00d+00  3.33d+00  3.55d+00  3.55d+00
    1.60d-08    0.00d+00  3.17d+00  3.34d+00  3.35d+00
    1.80d-08    0.00d+00  2.59d+00  2.73d+00  2.73d+00
    2.00d-08    0.00d+00  2.12d+00  2.21d+00  2.22d+00
```

External (analytical) evaluation of nonlinear capacitors can be performed by the following subroutine:

```
      SUBROUTINE SPUNCE (ID,IDF,VARG,NARG,VPAR,NPAR,VAL,MOPR)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
```

```
      DIMENSION VARG(NARG),NPAR(1),VPAR(1)
C ... exponential function
      IF (IDF.EQ.9) THEN
        VEXP=DEXP(-VARG(2)*VPAR(3))
        IF (MOPR.GE.0) THEN
C ....... find the capacitance
          VAL=VPAR(1)*(1.0-VPAR(2)*VEXP)
        ELSE
C ....... find the charge
          VAL=VPAR(1)*(VARG(2)+VPAR(2)*(VEXP-1.0)/VPAR(3))
        ENDIF
      ELSE
C ..... other evaluation functions
      ENDIF
      RETURN
      END
```

8. Enhanced circuit simulation

In enhanced circuit simulation, user-supplied routines augment the "standard" simulation capabilities of the package. Elements of enhanced circuit simulation are presented for time-domain analysis, one of the of the most useful, but computationally most complex, tasks of circuit simulation [Vlach]. The detailed algorithm depends upon the integration method used, but generally the simulation interval is divided into (usually variable) timesteps (sometimes called internal timesteps), and at each timepoint, the information from previous timepoints is used to derive the solution at the new timepoint. The stability and accuracy of the integration method has a significant effect on the stability, accuracy, and efficiency of the resulting simulation.

SPICE-PAC, similarly to SPICE, uses two most popular implicit integration methods [Cohen], the trapezoidal rule (default) and variable order backward differentiation (or Gear's) method, the second designed specifically to deal with stiff differential equations [Vlach].

In order to allow a flexible control of time-domain (transient) analysis, an in particular, to allow "external", user-defined control of this analysis, the original implementation has been modified in such a way that after each successful solution of an (internal) timestep, the subroutine SPURTR is invoked with the timepoint solution passed as one of parameters.

The SPURTR subroutine must be defined with the following (Fortran) header:

```
      SUBROUTINE SPURTR (VOUT,NOUT,TDEL,IRET)
      DOUBLE PRECISION VOUT(NOUT),TDEL
      INTEGER NOUT,IRET
```

where `VOUT` is an array which contains the values of outputs obtained for the present timepoint; VOUT(1) is always the value of simulated time, VOUT(2) is the value of the first output variable, etc.; `NOUT` is the length of `VOUT`; `TDEL` is the value of timestep, and `IRET` is an entry/return flag; on entry, IRET=–1 indicates the initial call, IRET=0 indicates an accepted timepoint solution, and IRET=+1 indicates nonconvergence; on exit, IRET=0 indicates continuation of analysis, IRET=+1 terminates transient analysis at the current timepoint, IRET=–2 turns off external control (i.e., SPURTR will not be called but the analysis continues), and IRET=–3 rejects the present timepoint, reduces the timestep four times, and resumes the analysis.

In the following example [Ztdom], the SPURTR routine is used to find the "exact" time instances at which the output signal of a simple MOSFET inverter circuit reaches the 10% and 90% levels.

```
** double inverter - transient analysis
.TR 1NS 20NS
.PRINT TRAN V(1) V(3) V(2)
VDD 5 0 DC=5
VIN 1 0 PULSE(0 5 1NS 2NS 2NS 3NS 10NS)
M1 5 1 2 5 PMOD L=3U W=6U AS=36P AD=36P
M2 2 1 0 0 NMOD L=3U W=3U AS=18P AD=18P
M3 5 2 3 5 PMOD L=3U W=6U AS=36P AD=36P
M4 3 2 0 0 NMOD L=3U W=3U AS=18P AD=18P
*... NMOS transistor model
.MODEL NMOD NMOS LEVEL=2
*... PMOS transistor model
.MODEL PMOD PMOS LEVEL=2
.END

... time :  2.0866d-09 output rising at 10%
... time :  2.7116d-09 output rising at 90%
... time :  6.9168d-09 output falling at 90%
... time :  7.3557d-09 output falling at 10%
... time :  1.2087d-08 output rising at 10%
... time :  1.2712d-08 output rising at 90%
... time :  1.6917d-08 output falling at 90%
... time :  1.7356d-08 output falling at 10%
```

The SPURTR routine uses the simplest form of linear interpolation to determine the intercept points (a more rigorous approach would iterate the intercept points using the "backtrack" mechanism with IRET=–3):

```
      SUBROUTINE SPURTR (VOUT,NOUT,TDEL,IRET)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION VOUT(NOUT)
```

```
      CHARACTER*21 TXT
      IF (IRET.LT.0) THEN
        TOLD=VOUT(1)
        VOLD=VOUT(3)
      ELSE IF (IRET.EQ.0) THEN
        TNEW=VOUT(1)
        VNEW=VOUT(3)
        TXT=' '
        IF (VOLD.LT.0.5 .AND. VNEW.GE.0.5) THEN
          TEMP=TOLD+(0.5-VOLD)*(TNEW-TOLD)/(VNEW-VOLD)
          TXT='output rising at 10%'
        ELSE IF (VOLD.LT.4.5 .AND. VOUT(3).GE.4.5) THEN
          TEMP=TOLD+(4.5-VOLD)*(TNEW-TOLD)/(VNEW-VOLD)
          TXT='output rising at 90%'
        ELSE IF (VOLD.GE.4.5 .AND. VNEW.LT.4.5) THEN
          TEMP=TOLD+(4.5-VOLD)*(TNEW-TOLD)/(VNEW-VOLD)
          TXT='output falling at 90%'
        ELSE IF (VOLD.GE.0.5 .AND. VNEW.LT.0.5) THEN
          TEMP=TOLD+(0.5-VOLD)*(TNEW-TOLD)/(VNEW-VOLD)
          TXT='output falling at 10%'
        ENDIF
        IF (TXT.NE.' ') WRITE(*,100) TEMP,TXT
  100   FORMAT(' ... time :',1PD12.4,1X,A)
        VOLD=VNEW
        TOLD=TNEW
      ENDIF
      RETURN
      END
```

External control of time-domain analysis is one of the basic mechanisms used in integrated mixed, analog-digital simulation [Goer,deMan] in which two different simulation techniques, the integration scheme of analog simulation and the event-driven digital simulation must be synchronized at selected timepoints.

9. Mixed-mode simulation

Simulation of mixed, analog-digital circuits is usually referred to as mixed-mode simulation [Goer,deMan]. A basic analog-digital interface has been implemented in SPICE-PAC [AllZ] to provide a table-driven conversion of analog to (multivalued) digital signals and vice versa. Piecewise linear characteristics of independent voltage and/or current sources [Vlad] are used for interactions between digital and analog subnetworks; the "smoothing" of discrete digital signals is thus implemented by piecewise linear functions. The interface is composed of two sections, one for analog-to-digital communication, and the second for

communication in the opposite direction. In the input (circuit specification) language, these two sections are described by two new directives, PUTLIST and GETLIST, respectively:

```
.PUTLIST:Tname1 Voutput1,Voutput2,...
.GETLIST:Tname1:Tname2 Vsource1,Vsource2,...
```

where `Tname1` is the name of a TABLE pseudoelement that defines the conversion table for analog-to-digital (and digital-to-analog) interface; `Tname2` is the name of another TABLE pseudoelement that defines the delay table for digital-to-analog conversion; each `Voutput` is a voltage output in the SPICE sense, i.e., it is either "V(node1,node2)" or "V(node1)" if the second node is zero; each `Vsource` is the name of an independent voltage source with a piecewise linear time-dependent function.

The conversion table is defined as an ordered sequence of increasing (threshold) voltages interposed with (internal) values of corresponding digital signals:

```
.TABLE Tname Volt0 Num1 Volt1 Num2 Volt2 Num3 ... Numk Voltk
```

Voltages in the range `Volt0` to `Volt1` are converted into a digital signal represented by the value `Num1`, etc. For digital-to-analog conversions, the extreme values `Num1` and `Numk` are translated into `Volt0` and `Voltk`, respectively, while all intermediate values are converted into "median" voltages, i.e., `Num2` corresponds to (Volt1+Volt2)/2, etc.

Presently, the delay table contains just three parameters, the delay time of the converted digital-to-analog signals (indicated in GETLIST), the rise rate (i.e., the rise time per 1V) and the fall rate.

In the following example, the digital part (shown later) is simply a two-input AND gate, with inputs and outputs described by the PUTLIST and GETLIST, respectively. There are two different conversion tables for analog-to-digital and digital-to-analog conversions, and also the rise and fall rates are different. The simulation results are shown in Fig.1:

```
VV 1 0 PULSE(-5.0,+5.0,0.5US,10NS,10NS,2US,5US)
R1 1 2 1K
C1 2 0 1NF
R2 1 3 1K
C2 3 0 200PF
VX 5 0 PWL(0 -5.0,15U -5.0)
RX 5 0 1K
.TRAN 50NS 10US
.PRINT TR V(2) V(3) V(5)
.PUTLIST:TCONV1 V(2),V(3)
.GETLIST:TCONV2:TDEL VX
.TABLE TCONV1 (-5.0,-1,-1.0,1,+5.0)
.TABLE TCONV2 (-5.0,-1,+1.0,1,+5.0)
.TABLE TDEL (1E-6,1E-7,5E-8)
.END
```

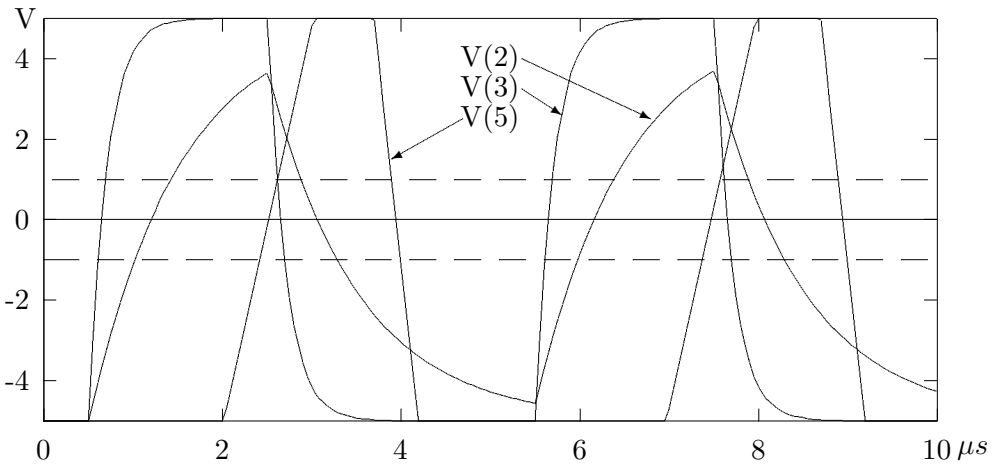


Fig.1. Mixed-mode simulation results.

In the SPICE-PAC's implementation of time-domain analysis (section 5), the logical condition `mixed_simulation` is "true" if both PUTLIST and GETLIST are nonempty, and then the analog-digital interfacing routine (SPPSIM) is invoked for each successfully solved timepoint. The interfacing routine performs analog-to-digital conversion of all PUTLIST voltages, and then checks:

- if any digital value created during this conversion differs from the corresponding (digital) value created in the previous invocation, and if all digital values remain the same
- if the present timepoint has been explicitly requested by the external simulation routines (for example, because of the "internal" timing mechanisms).

If both checks fail, the interfacing routine returns and the time-domain analysis continues. If the present timepoint is requested by the digital simulation (second condition), the routine SPUSIM is invoked to perform the simulation of the digital part (at the gate, functional or behavioral level) for the present timepoint. In the case of the first condition satisfied, i.e., if there is at least one "new" digital value after the conversion, before an invocation of SPUSIM the timepoint is (iteratively) adjusted to a value corresponding to the closest conversion threshold. After completion of digital simulation, the digital-to-analog conversions are performed for those (digital) signals which are indicated in the GETLIST specifications, and which changed their (digital) values during the invoked digital simulation. Then the (analog) time-domain simulation resumes.

The SPUSIM routine is either a simple interfacing routine to a "standard" logic simulation program, or a user-supplied routine (which may "drive" or enhance a logic simulator). If it is a user routine, it must be defined in a way consistent with the following (FORTRAN) header:

```
SUBROUTINE SPUSIM (TIME,LINP,NINP,LOUT,NOUT,MARK)
DOUBLE PRECISION TIME
INTEGER LINP(NINP),LOUT(NOUT),MARK
```

where the parameters are:

TIME - the value of the (simulated) time,

LINP - an array of length NINP which contains the coverted values of PUTLIST data,

NINP - the length of LINP, i.e., the number of analog-to-digital signals,

LOUT - an array of length NOUT which return the new (digital) values of GETLIST variables; on entry LOUT contains "previous" values of GETLIST variables, so only "changes" need to be stored in LOUT,

NOUT - the length of LOUT, i.e, the number of digital-to-analog signals,

MARK - an entry/return flag; on entry: MARK=–1 indicates the initial invocation, MARK=0 indicates an accepted timepoint (i.e., a "regular" invocation), while non-convergence (and termination of analog simulation) is indicated by MARK=+1; on exit: MARK=0 indicates continuation of analysis, and MARK=+1 request to terminate the analysis at the current timepoint.

The following example of the SPUSIM routine simulates the "logic" AND gate used in the previous example:

```
      SUBROUTINE SPUSIM (TIME,LINP,NINP,LOUT,NOUT,MARK)
      DOUBLE PRECISION TIME
      DIMENSION LINP(NINP),LOUT(NOUT)
C  It is assumed that the internal representation of  "1"  is greater
C  than that of "0"; then the logical AND corresponds to the standard
C  MIN (or MIN0 for INTEGER arguments) function.
      LOUT(1)=MIN0(LINP(1),LINP(2))
      RETURN
      END
```

Mixed-mode simulation offers a significant reduction of simulation times with respect to "all analog simulation" (in the range of one to two orders of magnitude depending on the level of digital simulation, behavioral, functional or gate-level). However, the development process of such simulators is quite unreliable and time-consuming because the simulation code (SPUSIM routines) must be developed for each application, and this code must be tested and validated before actual simulations take place. Therefore, a "better" approach is needed, in which the digital simulation is derived directly from design specifications by appropriate extensions of the specification language, circuit description processors and evaluation routines.

10. Simple postprocessor

A postprocessor has been added to SPICE-PAC in order to provide an elementary capability for processing the results of circuit analyses; evaluation of sensitivities by the perturbation method is a good example of postprocessing.

The postprocessor operates on results of "group" analyses, i.e., analyses performed repeatedly for a given set of parameter values (the perturbations of circuit elements for sensitivity evaluations correspond to such a set of parameter values). Group analyses are indicated by an extended version of SPICE PAC's ".var" operation that specifies the values of (static) circuit variables (defined in the extended circuit description). For two variables, say a (parasitic) resistor "Rpar" and capacitor "Cpar", defined as the first two circuit variables:

```
 .....
.END/EXT
.VAR Rpar
.VAR Cpar
.END
```

the set of five different combinations of parameter values (i.e., the "nominal values", and "positive" and "negative" perturbations for each "Rpar" and "Cpar") can be defined as:

```
.var{1,2}=({100,10E-12},{105,10E-12},{95,10E-12},{100,11E-12},{100,9E-12})
```

The perturbations of "Rpar" are by 5% and those of "Cpar" by 10%. The subsequent analysis (for example, ".ac") will create five different variants of results, corresponding to consecutive combinations of parameter values from this ".var" list. The postprocessor operates on such collections of results.

The general idea of describing the postprocessor operations is to define a list of new, computed "outputs" that can be displayed, printed or stored in a file. The postprocessor's specification is thus a sequence of new output definitions (the postprocessor is invoked by the ".pproc" operation in the "standard" SPICE-PAC's driver):

```
.pproc(output1,output2,...)
```

where each `output` definition is an expression composed of arithmetic operators, constants and "#variant:column" result identifiers, with "#variant" denoting a variant of results (i.e., results corresponding to a particular combination of parameter values) and ":column" denoting a particular output variable of the results "#variant". There also is a "composition" operator "_" (underscore) which combines two (real) output variables into a single complex variable, as required in frequency-domain calculations. This operator has the highest priority, so it binds stronger than any other arithmetic operator.

Normalized AC sensitivities with respect to RR and CC in the following simple RC circuit:

```
VV 1 0 AC=1
RR 1 2 1K
CC 2 0 1NF
.AC 1K,10K,100K
.PRINT AC VR(2) VI(2)
.END/EXT
.VAR RR
.VAR CC
.END
```

can thus be obtained by repeated analyses with consecutive perturbations of RR and CC and then postprocessing:

```
.var{1,2}=({1000,1D-9},{1050,1D-9},{950,1D-9},{1D3,0.95D-9},{1D3,1.05D-9})
.ac
.pproc(10.0*(#3:1_#3:2-#2:1_#2:2)/(#1:1_#1:2),
       10.0*(#5:1_#5:2 #4:1_#4:2)/(#1:1_#1:2))
```

(observe that the perturbations of RR have been reversed to obtain the effect of sensitivity with respect to conductance rather than resistance), which produces the following results (each complex expression is equivalent to two consecutive "outputs"; so "#1" and "#2" are the real and imaginary parts of a (complex) sensitivity with respect to RR, etc.):

```
   FREQ        #1        #2        #3        #4
1.00d+03   3.92d-05  6.28d-03  3.92d-05  6.28d-03
1.00d+04   3.93d-03  6.26d-02  3.93d-03  6.26d-02
1.00d+05   2.83d-01  4.51d-01  2.83d-01  4.51d-01
```

The results are practically the same as the "exact" results obtained for the small signal AC sensitivities.

There are several implementation restrictions in the present version of the postprocessor, e.g., the maximum number of operators, constants and arguments, or the maximum number of defined outputs, however, all these restrictions can easily be changed by appropriate modifications of the postprocessor's code ("t1post.f" file in the standard distribution).

R e f e r e n c e s

[AllZ] P.E. Allen, W.M. Zuberek, "Simulation of mixed, analog-digital circuits with SPICE-like simulators" (paper submitted for publication).

[Cohen] E. Cohen, "Program reference for SPICE 2"; Memorandum UCB/ERL M592, University of California, Berkeley CA 94720, 1976.

[Goer] R. Goering, "A full range of solutions emerge to handle mixed-mode simulation"; Computer Design, vol.27, no.3, pp.57-65, 1988.

[deMan] H. de Man, G. Arnout, P. Reynaert, "Mixed-mode simulation techniques and their implementation in DIANA"; in: "Computer Design Aids for VLSI Circuits", P. Antognetti, D.O. Pederson, H. de Man (eds.), Sijthoff and Noordhoff 1981.

[McCal] W.J. McCalla, "Fundamentals of computer-aided circuit simulation"; Kluwer Academic Publ. 1988.

[Peder] D.O. Pederson, "Computer aids in integrated circuit design"; in: "Computer Design Aids for VLSI Circuits", P. Antognetti, D.O. Pederson, H. de Man (eds), Sijthoff and Noordhoff 1981.

[Poiv] Ch. Poivey, "Methodes d'optimisation globale pour la CAO de circuits integres; interface avec le simulateur SPICE-PAC" (Global optimization methods for CAD of integrated circuits; an interface to the SPICE-PAC simulation package); These de Docteur Ingenieur, l'Universite Blaise Pascal, serie D.I., no.203, Clermont-Ferrant, France, 1987.

[Vlach] J. Vlach, K. Singhal, "Computer methods for circuit analysis and design"; Van Nostrand Reinhold 1983.

[Vlad] A. Vladimirescu, K. Zhang, A.R. Newton, D.O. Pederson, A.L. Sangiovanni-Vincentelli, "SPICE Version 2G - User's Guide "; Department of Electrical Engineering and Computer Sciences, University of California, Berkeley CA 94720, 1981.

[Zlib] W.M. Zuberek, "SPICE-PAC and libraries of standard modules"; Proc. Canadian Conf. on VLSI (CCVLSI-84), Edmonton, Canada, 1984.

[Zopt] W.M. Zuberek, "Reverse and indirect communication in interfacing circuit simulation with optimization"; Proc. CIPS Congress 86, Vancouver, Canada, pp.103-108, 1986.

[Zpar] W.M. Zuberek, "Parameterized subcircuits in the SPICE-PAC package of simulation subroutines"; Proc. Canadian Conference on VLSI (CCVLSI-87), Winnipeg, Canada, pp.117-122, 1987.

[Zsour] M.S. Zuberek, W.M. Zuberek, "Enhanced controlled sources as device models in the SPICE-PAC simulation package"; Proc. 30-th Midwest Symp. on Circuits and Systems, Syracuse NY, pp.603-606, North-Holland 1988.

[Ztdom] W.M. Zuberek, "Dynamic control of time-domain analysis in the SPICE-PAC simulation package"; Proc. 22-nd Asilomar Conf. on Circuits, Signals and Computers, Pacific Grove CA (in print).

[Zvar] M.S. Zuberek, W.M. Zuberek, "Aggregate circuit variables in the SPICE-PAC simulation package"; Proc. 31-st Midwest Symp. on Circuits and Systems, St. Louis MO (in print).

APPENDIX A

SPICE-PAC main (or interfacing) subroutines

SPICEA	initializes the package and reads circuit description,
SPICEB	defines circuit variables,
SPICEC	sets internal structures and performs initial processing,
SPICED	defines parameters for DC analysis,
SPICEE	defines execution-time limit,
SPICEF	defines frequencies for AC, NOISE and DISTORTION analyses,
SPICEG	defines parameters for DISTORTION analysis,
SPICEH	defines parameters for FOURIER analysis,
SPICEI	defines initial conditions (as node and/or device voltages),
SPICEJ	sets and resets internal flags,
SPICEK	defines parameters for DC TRANSFER FUNCTION analysis,
SPICEL	activates definitions of parameters and outputs,
SPICEM	defines the temperature for subsequent analyses,
SPICEN	defines parameters for NOISE analysis,
SPICEO	defines outputs for different analyses,
SPICEP	determines internal pointers for circuit element names,
SPICEQ	defines output variables,
SPICER	performs DC, TRANSIENT, AC, NOISE, DISTORTION, FOURIER, DC TRANSFER FUNCTION and AC sensitivity analyses, and DC OP-POINT solution,
SPICES	performs DC SENSITIVITY analysis,
SPICET	defines parameters for TRANSIENT analysis,
SPICEU	updates circuit variables,
SPICEV	retrieves actual values of circuit variables,
SPICEW	retrieves SPICE-PAC execution times,
SPICEX	defines parameters and outputs using symbolic form,
SPICEY	retrieves the names of output variables, circuit elements and circuit variables.

APPENDIX B

SPICE-PAC's parameters of circuit elements

Nonlinear capacitors and inductors:

#n	coefficient "n" of the polynomial function (n=0,1,...)

Transmission lines:

ZO	characteristic impedance
TD	transmission delay
ICV1	initial voltage at the input port for the TRANSIENT analysis
ICI1	initial current at the input port for the TRANSIENT analysis
ICV2	initial voltage at the output port for the TRANSIENT analysis
ICI2	initial current at the output port for the TRANSIENT analysis

Dependent (nonlinear) voltage and current sources:

#n	coefficient "n" of the polynomial function (n=0,1,...)

Independent (voltage and current) source parameters:

DC	DC voltage/current
ACM	magnitude of AC voltage/current
ACP	phase of AC voltage/current
#n	parameter "n" of time-dependent source function (n=0,1,...)

Diode parameters:

AREA	area factor
IQVD	initial voltage for the OP-POINT solution
ICVD	initial voltage for the TRANSIENT analysis

Diode model parameters:

IS	saturation current
RS	ohmic resistance
N	emission coefficient
TT	transit time
CJO	zero-bias junction capacitance
VJ	junction potential

M	grading coefficient
EG	energy gap
XTI	saturation current temperature exponent
KF	flicker noise coefficient
AF	flicker noise exponent
FC	forward-bias nonideal junction capacitance coefficient
BV	reverse breakdown voltage
IBV	current at breakdown voltage

Diode noise components:

RS	ohmic resistance
ID	diode current
FN	flicker noise

BJT parameters:

AREA	area factor
IQVBE	initial VBE voltage for the OP-POINT solution
ICVBE	initial VBE voltage for the TRANSIENT analysis
IQVCE	initial VCE voltage for the OP-POINT solution
ICVCE	initial VCE voltage for the TRANSIENT analysis

BJT model parameters:

IS	saturation current
BF	ideal forward current gain
NF	forward current emission coefficient
VAF	forward Early voltage
IKF	forward knee current
ISE	base-emitter leakage saturation current
NE	nonideal low-current base-emitter emission coefficient
BR	ideal reverse current gain
NR	reverse current emission coefficient
VAR	reverse Early voltage
IKR	reverse knee current
ISC	base-collector leakage saturation current
NC	nonideal low-current base-collector emission coefficient
RB	base ohmic resistance
IRB	current at which base resistance decreases halfway
RBM	minimum base resistance at high currents
RE	emitter ohmic resistance
RC	collector ohmic resistance

CJE	zero-bias base-emitter junction capacitance
VJE	base-emitter built-in potential
MJE	base-emitter junction exponential factor
TF	forward transit time
XTF	coefficient for bias dependence of forward transit time
VTF	voltage for bias dependence of forward transit time
ITF	high-current parameter for effect on forward transit time
PTF	excess phase at frequency 1/(2*PI*TF) Hz
CJC	zero-bias base-collector junction capacitance
TR	reverse transit time
CJS	collector-substrate capacitance
VJS	substrate junction built-in potential
MJS	substrate junction exponential factor
XTB	forward and reverse current gain temperature coefficient
EG	energy gap
XTI	saturation current temperature exponent
KF	flicker noise coefficient
AF	flicker noise exponent
FC	forward-bias nonideal junction capacitance coefficient

BJT noise components:

RB	base ohmic resistance
RC	collector ohmic resistance
RE	emitter ohmic resistance
IB	base current
IC	collector current
FN	flicker noise

JFET parameters:

AREA	area factor
IQVDS	initial VDS voltage for the OP-POINT solution
ICVDS	initial VDS voltage for the TRANSIENT analysis
IQVGS	initial VGS voltage for the OP-POINT solution
ICVGS	initial VGS voltage for the TRANSIENT analysis

JFET model parameters:

VTO	threshold voltage
BETA	transconductance parameter
LAMBDA	channel length modulation parameter
RD	drain ohmic resistance

RS	source ohmic resistance
CGS	zero-bias gate-source junction capacitance
CGD	zero-bias gate-drain junction capacitance
PB	gate junction potential
IS	gate junction saturation current
KF	flicker noise coefficient
AF	flicker noise exponent
FC	forward-bias nonideal junction capacitance coefficient

JFET noise components:

RD	drain ohmic resistance
RS	source ohmic resistance
ID	drain current
FN	flicker noise

MOSFET parameters:

W	channel width
L	channel length
AD	area of the drain diffusion
AS	area of the source diffusion
PD	perimeter of the drain junction
PS	perimeter of the source junction
NRD	number of squares of the drain diffusion
NRS	number of squares of the source diffusion
IQVDS	initial VDS voltage for the OP-POINT solution
ICVDS	initial VDS voltage for the TRANSIENT analysis
IQVGS	initial VGS voltage for the OP-POINT solution
ICVGS	initial VGS voltage for the TRANSIENT analysis
IQVBS	initial VBS voltage for the OP-POINT solution
ICVBS	initial VBS voltage for the TRANSIENT analysis

MOSFET model parameters:

VTO	zero-bias threshold voltage
KP	intrinsic transconductance parameter
GAMMA	bulk threshold parameter
PHI	surface potential at strong inversion
LAMBDA	channel length modulation parameter
RD	drain ohmic resistance
RS	source ohmic resistance
CBD	zero-bias bulk-drain junction capacitance

CBS	zero-bias bulk-source junction capacitance
IS	bulk junction saturation current
PB	bulk junction potential
CGSO	gate-source overlap capacitance
CGDO	gate-drain overlap capacitance
CGBO	gate-bulk overlap capacitance
RSH	drain and source diffusion sheet resistance
CJ	zero-bias bulk junction bottom capacitance
MJ	bulk junction bottom grading coefficient
CJSW	zero-bias bulk junction sidewall capacitance
MJSW	bulk junction sidewall grading coefficient
JS	bulk junction reverse saturation current
TOX	oxide thickness
NSUB	effective substrate doping
NSS	effective surface state density
NFS	effective fast surface state density
TPG	type of gate material
XJ	metalurgical junction depth
LD	lateral difussion coefficient
UO	surface mobility
UCRIT	critical field for mobility degradation
UEXP	critical field exponent in mobility degradation
UTRA	transverse field coefficient (mobility)
VMAX	maximum drift velocity of carriers
NEFF	total channel charge (fixed and mobile) coefficient
XQC	coefficient of channel charge share attributed to drain
KF	flicker noise coefficient
AF	flicker noise exponent
FC	forward-bias nonideal junction capacitance coefficient
DELTA	width effect on threshold voltage
THETA	mobility modulation
ETA	static feedback
KAPPA	saturation field factor

MOSFET noise components:

RD	drain ohmic resistance
RS	source ohmic resistance
ID	drain current
FN	flicker noise