First Int. Conf. on Computer Technology, Systems and Applications; Hamburg, Germany, May 11-15, 1987, pp.229-232.
Copyright © 1987 IEEE (DOI 10.10-7/08186-07734).

CIRCUIT DECOMPILATION IN THE SPICE-PAC PACKAGE OF SIMULATION SUBROUTINES

W.M. Zuberek

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

Abstract

SPICE-PAC is a package of simulation subroutines obtained by redesigning the popular SPICE circuit simulator from University of California at Berkeley. The package is compatible with the SPICE program but also provides several extensions, e.g., it supports circuit variables, contains an interface to hierarchical libraries of QstandardU designs, etc. Recently, the package has been extended by Qdecompilation U procedures to provide symbolic description of circuits refined during interactive analyses, or modified by circuit optimization programs.

1. INTRODUCTION

Computer-aided circuit analysis or circuit simulation is a widely accepted tool in the area of integrated circuit design. By using this method, circuit designers can easily explore the effects of different designs on circuit performance. The SPICE program [2,7,8] developed at the University of California at Berkeley has become one of the most popular "second-generation" circuit simulators.

"Second-generation" simulation programs are usually batch-oriented, which means that they require a new, independent run for each modification of the analyzed circuit. This is too restrictive in application based on repeated circuit simulations, for example interactive simulation or circuit optimization [1,3,4,5]. Therefore a more flexible structure for circuit simulators is needed in which different analyses can be performed selectively, and which provides access to the internal representation of circuit elements in order to modify their values. The simulators should have the structure of a set (or a package) of subroutines rather than a program with one, fixed sequence of operations. SPICE-PAC [9,10] is such a package of simulation subroutines, derived from the SPICE-2G.6 circuit simulator.

SPICE-PAC accepts circuit descriptions in the same form as the original SPICE-2G.6 program with only a few minor exceptions. The package provides [9,11,12]:

- all the analyses available in the SPICE 2G programs,
- a hierarchical naming scheme for (nested) subcircuits;
- access to circuit variables as required in interactive simulation and circuit optimization;
- dynamic definitions of parameters and declarations of outputs for all analyses,
- parameterized subcircuit invocations; during subcircuit expansion, subcircuits can be adjusted to the actual needs by substituting subcircuit element values indicated in a parameter-list,
- an interface to hierarchical libraries of standard modules (or "building blocks"); library modules correspond to standard SPICE subcircuits stored in individual files within a file system, and included into a circuit by (parameterized) module invocations,
- an extended circuit description which can contain predefinitions of parameters, outputs and variables used in circuit analyses.

SPICE-PAC contains 26 main subroutines (and approximately 250 internal subroutines and functions), but it does not provide the main program which must be supplied by the user to "drive" the subroutines, i.e., to call the subroutines which read and process (source) circuit descriptions, define circuit variables, perform analyses, etc., as required by a particular application (simple interactive drivers are usually distributed with the package). The user can also supply his own versions of selected subroutines [11] and replace the corresponding SPICE algorithms enhancing the original SPICE (and SPICE-PAC) facilities.

One of recent extensions of the package is an interface to decompilation procedures which convert SPICE-PAC internal representation of circuits (a multilevel list structure with expanded subcircuits, library modules, etc.) into a symbolic description used as input by SPICE (and SPICE-PAC). This provides a simple way to obtain a familiar source description of circuits modified during interactive analyses or refined by optimization programs. Moreover, since the source description is usually much more compact than the corresponding (expanded) internal representation, it is more convenient for storing (in files or databases). Decompilation facilities can also be used in migration of circuit descriptions from one design system to another, when a conversion of source descriptions is required because of different input languages.

The paper briefly describes the original list-structured internal representation of circuit descriptions in SPICE-like programs, indicates the modifications of the original structures required by decompilation routines, and outlines the decompilation process. A simple example of circuit description compiled and decompiled by SPICE-PAC is included as an illustration of described capabilities.

2. INTERNAL CIRCUIT REPRESENTATION

During processing of the source circuit description, the input module of SPICE-like programs organizes all circuit elements of the same class (resistors, capacitors, etc.) into list structures composed of element descriptors (Fig.1). The descriptors store all attributes (numerical, e.g., the resistance of a resistor and its temperature coefficients, and textual, e.g., the name of an element) associated with a single circuit element. They also contain internal pointers to related data stored in other descriptors, auxiliary tables, etc.

There are four types of circuit element descriptors used in internal representation of circuits (Fig.1):

simple descriptors (e.g., resistors, transmission lines, models of semiconductor devices); all attributes and related information are organized in a single descriptor (however descriptors for different classes of circuit elements may have different sizes), extended descriptors (e.g., nonlinear capacitors or inductors, independent and dependent voltage and current sources, subcircuit invocations); some attributes are stored in auxiliary tables which are indicated by corresponding pointers; the size of these tables may vary from element to element (e.g., polynomials describing nonlinear capacitances of different capacitors may have different degrees, i.e., different numbers of coefficients stored in a table); extended descriptors may indicate several auxiliary tables (e.g., dependent voltage and current sources use one auxiliary table to indicate consecutive (variable) sources, and another table to store coefficients of the dependency polynomial),

linked descriptors (e.g., semiconductor devices and their models, subcircuit invocations and corresponding subcircuit definitions); each device descriptor indicates the descriptor of a model associated with this device; model descriptors are rather complex and contain numerous parameters [2] but usually several devices are associated with the same model and consequently model parameters need to be specified just once,

list descriptors (e.g., subcircuit definitions); there is a list of internal simple or extended descriptors (subcircuit elements) associated with a basic descriptor (subcircuit definition); subcircuit elements do not belong to the list structure representing the circuit, however, during processing of source descriptions, subcircuit definitions corresponding to subcircuit invocations are expanded by copying and inserting definition lists into the main linked element lists; at the same time the definition node numbers are converted into unique circuit node numbers, and subcircuit parameter substitutions are performed.

Headers of list structures for all classes of circuit elements are organized into a vector of pointers (CELists in Fig.1) which is stored in one of global areas, i.e., areas available to many routines and modules of SPICE-like programs. Initial elements of this vector correspond to:

1	R	resistors
2	C	capacitors
3	L	inductors
4	M	mutual inductors
5	G	voltage controlled current sources
6	E	voltage controlled voltage sources
7	F	current controlled current sources
8	H	current controlled voltage sources
9	V	independent voltage sources
10	I	independent current sources
11	D	semiconductor diodes
12	Q	bipolar junction transistors
13	J	junction-field-effect transistors
14	M	MOS field-effect transistors
17	T	transmission lines
19	X	subcircuit invocations
20	*	subcircuit definitions
21	*	diode models
22	*	BJT models
23	*	JFET models
24	*	MOSFET models

There is an additional pointer in each descriptor which indicates whether t he descriptor belongs to an element at the main (or top) level of circuit description (the pointer is "nil"), or it has been created as a result of subcircuit expansion, and then the pointer indicates the corresponding descriptor in the list of subcircuit invocations (CEList[19]). By searching lists of circuit elements (CEList), it is thus possible to trace all circuit elements which correspond to a particular subcircuit invocation and expansion (and there may be many different invocations of the same subcircuit definition).

List structures provide a very simple access to all circuit elements of the same class which is quite important for efficient organization of subsequent numerical evaluations.

3. HIERARCHICAL SUBCIRCUITS

SPICE-like programs support hierarchical subcircuits which means that subcircuit definitions may contain invocations of (other) subcircuits. In the SPICE-PAC package, subcircuit invocations may contain lists of (subcircuit) parameters which are sequences of simple substitutions of the form *attributename* = *value*, used to replace the original subcircuit element attributes by the values indicated in parameter lists. Attributes indicated in parameter lists may refer to nested subcircuits by using qualified names, where qualifiers (separated by dots ".") are pseudoelement names of subsequent nested subcircuit invocations. For example, "XX.X2.R1" denotes the resistance of the element R1 in the subcircuit (invoked by) X2 of the subcircuit (invoked by) XX.

The expansion of subcircuits is performed in a top-down manner which can be described by the following iteration:

```
pointer1 := CEList[19];
while pointer1 $\neq$ nil do
begin find corresponding subcircuit definition using
        the list of subcircuit definitions CEList[20];
    pointer2 := list of subcircuit elements;
    while pointer2 $\neq$ nil do
    begin using pointer2 copy the element into CEList;
        perform node mapping;
        perform subcircuit parameter substitutions;
        move pointer2 to the next element end;
    move pointer1 to the next element end;
```

It should be noticed that subcircuit invocations within subcircuit definitions are appended to the list CEList[19]; the iteration process must thus be dynamic since CEList[19] may extend during its processing. Moreover, subcircuit parameters indicating nested subcircuits are (temporarily) attached to the corresponding subcircuit invocations identified by the first (or top) qualifiers, and these top qualifiers are removed. Hierarchical subcircuit expansion is thus associated with passing (qualified) parameters to consecutive levels of nested subcircuits.

4. CIRCUIT DECOMPILATION

One of main design objectives for decompilation procedures was to generate the symbolic descriptions in a form as similar to the original description as possible. This required, however, a number of modifications in the internal SPICE representation if circuits in order to preserve some additional source information:

- the lists of nodes in subcircuit definitions and subcircuit invocations must be saved for decompilation process,

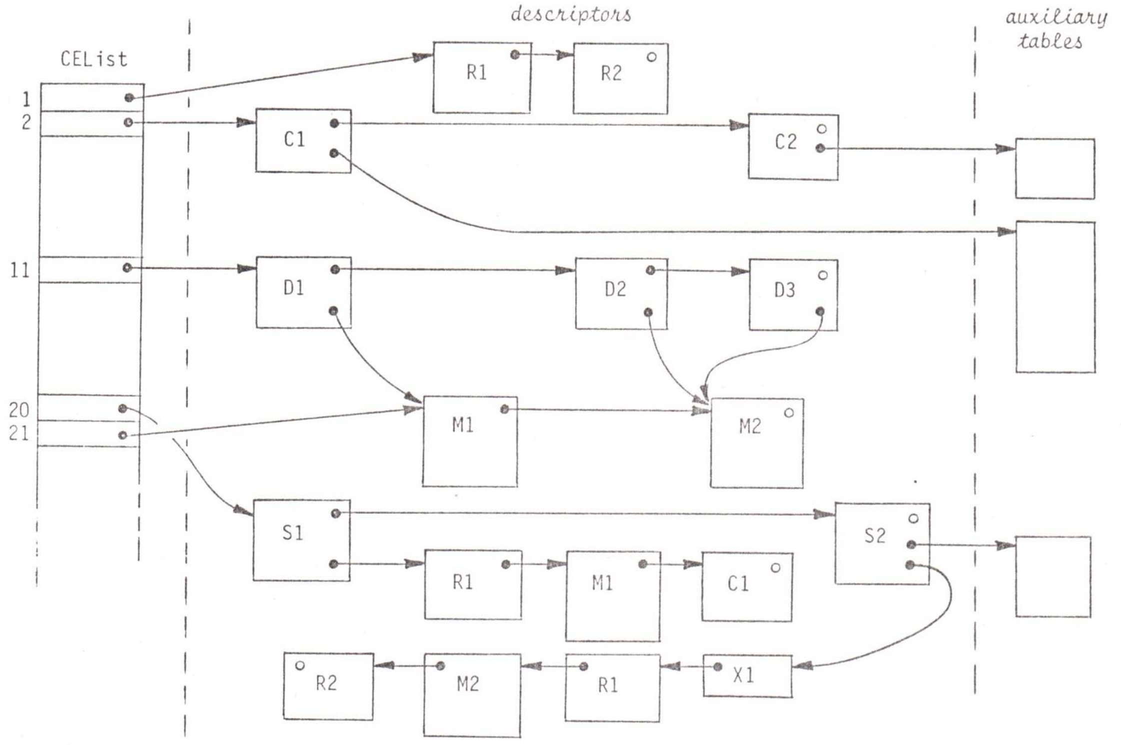

Fig.1. list structures for internal representation of circuits.

- the lists of original subcircuit parameters must be stored,
- the original values of preprocessed element attributes must be preserved for possible references; this is especially important in the case of device models since many model parameters are preprocessed irreversibly.

The decompilation is performed in a bottom-up way, i.e., it starts with subcircuit definitions (if there are any);

```
pointer1 := CEList[20];
while pointer1 ≠ nil do
begin pointer2 := list of subcircuit elements;
    while pointer2 ≠ nil do
    begin output the element indicated by pointer2;
        move pointer2 to the next element end;
    move pointer1 to the next element end;
```

Then follows the main circuit:

```
for i:=1 to 24 do
if i ≠ 20 then
begin pointer1 := CEList[i];
    while pointer1 ≠ nil do
    begin if pointer1 indicates a main level element then
        begin output the element indicated by pointer1;
            if i = 19 then
            begin checker(pointer1);
                output subcircuit parameters end
        end
    end;
    move pointer1 to the next element end;
```

and the recursive procedure "checker" compares subcircuit definitions with corresponding expanded elements, and generates (additional) subcircuit parameters for all detected differences; recursive invocations of "checker" correspond to nested subcircuit invocations:

```
procedure checker(pointer1);
begin find subcircuit definition for the invocation
        indicated by pointer1;
    pointer2 := list of subcircuit elements;
    while pointer2 ≠ nil do
    begin using CEList find the element obtained
        during subcircuit expansion from the element
        indicated by pointer2, and indicate it by pointer3;
        if pointer2 indicates subcircuit invocation then
            checker(pointer3)
        else compare attributes of elements indicated
            by pointer2 and pointer3, and generate
            subcircuit parameters for all attributes
            that are different end;
    move pointer2 to the next element end
end (* checker *);
```

As an example, a simple resistive voltage divider (Fig.2) is described as a circuit composed of nested subcircuits:

```
*** EXAMPLE - voltage divider
** subcircuit definitions
.SUBCKT DIV1 1,2,3
R1       1,2   1K
R2       2,3   1K
.ENDS
.SUBCKT DIV3 1,2,3
```

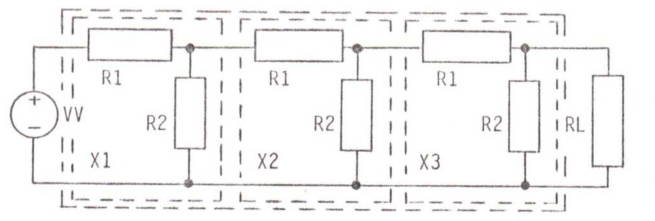

Fig.2. Circuit with nested subcircuits.

```
X1       1,4,3 DIV1
X2       4,5,3 DIV1
X3       5,2,3 DIV1
.ENDS
** main circuit
VV       1,0   5
XX       1,2,0 DIV3 (X3.R1=500)
RL       2,0   2K
.DC (VV,-5,+5,1)
.PRINT DC V(1),V(2)
.END
```

If this circuit is modified (for example interactively) in such a way that the values of two resistors, XX.X1.R2 and XX.X3.R1 (Fig.2), are replaced by 500 and 2000, respectively, the output created by decompilation procedures is as follows:

```
**** SPICE-PAC 2G6d.87.01 (MUN:X)   DATE : 15 JAN 87
** subcircuit definitions
.SUBCKT DIV1 1,2,3
R1       1,2   +1E+3
R2       2,3   +1E+3
.ENDS
.SUBCKT DIV3 1,2,3
X1       1,4,3   DIV1
X2       4,5,3   DIV1
X3       5,2,3   DIV1
.ENDS
** main circuit description
VV       1,0   +5
XX       1,2,0   DIV3 (X1.R2=+500,X3.R1=+2E+3)
RL       2,0   +2E+3
.DC (VV,-5,+5,+1)
.PRINT DC V(1),V(2)
.END
```

in which the modified values are represented by subcircuit parameters associated with the subcircuit invocation at the top level of circuit description.

5. CONCLUDING REMARKS

An implementation of decompilation capabilities (i.e., conversion of internal representation of circuits to its original source or symbolic form) in the SPICE-PAC package has been described. This provides a simple mechanism to obtain a refined original (symbolic) description of a circuit that has been subjected to modification during interactive analyses, or QtunedU by an optimization procedure. Decompilation facilities can also be used in conversions of different (source) circuit representations, i.e., conversions between input languages of different design and/or analysis systems.

It should be noticed that the description of internal structures used for representation of circuits is simplified and there are many other details and technicalities which have been neglected [2,9] since they do not influence the general ideas presented in this paper. Also, similar data structures are used in SPICE-3, the new version of SPICE, recently released by the University of California at Berkeley [6]. At the present time it is not quite clear if it will be feasible to transfer the SPICE-PAC enhancements to the SPICE-3 framework, but such a projects is being considered recently.

Decompilation capabilities described in this paper are implemented in the SPICE-PAC package versions 2G6d and beyond.

Acknowledgement

medskip The Natural Sciences and Engineering Research Council of Canada partially supported this research through Operating Grant A8222.

REFERENCES

[1] R.K. Brayton, G.D. Hachtel, A.L. Sangiovanni-Vincentelli, A survey of optimization techniques for integrated-circuit design; Proc. of the IEEE, vol.69, no.10, pp.1334-1362, 1981.

[2] E. Cohen, Program reference for SPICE-2; ERL Memorandum M520, Electronics Research Laboratory, University of California, Berkeley CA, 1976.

[3] J.K. Fidler, C. Nightingale, **Computer aided circuit design**; J. Wiley & Sons 1978.

[4] A.R. Newton, D.O. Pederson, A.L. Sangiovanni-Vincentelli, C.H. Sequin, Design aids for VLSI - The Berkeley perspective; IEEE Trans. Circuit and Systems, vol.28, no.7, pp.666-680, 1981.

[5] W.T. Nye, A.L. Tits, An enhanced methodology for interactive optimal design; Proc. IEEE International Symp. on Circuits and Systems, Newport Beach CA, 1983.

[6] T. Quarles, A.R. Newton, D.O. Pederson, A. Sangiovanni-Vincentelli, SPICE 3A7 User's Guide; Dept. of Electrical Engineering and Computer Science, University of California, Barkeley, CA 94720, 1986.

[7] D.O. Pederson, A historical review of circuit simulation; IEEE Trans. Circuits and Systems, vol.31, no.1, pp.103-111, 1984.

[8] A. Vladimirescu, K. Zhang, A.R. Newton, D.O. Pederson, A.L. Sangiovanni-Vincentelli, SPICE Version 2G - User's Guide (10 Aug. 1981); Department of Electrical Engineering and Computer Sciences, University of California, Berkeley CA 94720, 1981.

[9] W.M. Zuberek, SPICE-PAC 2G6a.84.05 - User's Guide; Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada A1C 5S7, Technical Report 8404, 1984.

[10] W.M. Zuberek, SPICE-PAC, a package of subroutines for interactive simulation and optimization of circuits; Proc. IEEE Int. Conf. on Computer Design, Port Chester NY, pp.492-496, 1984.

[11] W.M. Zuberek, P. Gillard, Enhanced circuit simulation using the SPICE-PAC package; Proc. 28 Midwest Symp. on Circuit and Systems, Louisville KY, pp.182-185, 1985.

[12] W.M. Zuberek, P. Gillard, Implementation of dynamic circuit variables in the SPICE-PAC package of simulation subroutines; Proc. 29 Midwest Symp. on Circuit and Systems, Lincoln NE, 1986.