9-th Int. Phoenix Conf. on Computers and Communications (PCCC'90), Scottsdale, AZ, 21-24 March 1990, pp.832-839.
Copyright © 1990 IEEE (DOI 10.1109/PCCC.1990.101706).

SOFTWARE INTERFACES FOR INTEGRATED SIMULATION APPLICATIONS

W.M. Zuberek

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

A b s t r a c t

In integrated simulation applications, simulation tools interact with other components of computer-aided design systems at the level of internal structures, sharing some internal data and communicating through procedural invocations. There are two basic types of interactions between interacting tools, the so called reverse and indirect communication methods. Reverse communication returns to the invocation environment after each required task; it is a rather flexible technique but it requires global control methods which can be quite unreliable and difficult for modifications in nontrivial applications. Indirect communication uses local control at different levels of hierarchical organization, passing global information from one level of the hierarchy to another; it corresponds thus to hierarchical structure of tools and to different levels of abstraction at different levels of hierarchy. The paper describes several integrated applications built around a circuit simulation environment, and discusses interfaces designed for the integration of different tools.

1. INTRODUCTION

Computer-aided circuit analysis, or circuit simulation has become a widely accepted tool in the area of integrated circuit design [BCSV, FiNi, NSST, Pede, VlSi]. By using this method, circuit designers can easily investigate the effects of different designs on the circuit performance. The process of selecting an appropriate design to satisfy the required specifications is usually based on consecutive approximations as well as on the designer's experience, and there is a clear need for simulators integrated with other computer-aided design tools such as optimization methods, mixed analog-digital simulators, mixed symbolic-numerical simulators, etc. [AnHu, LKCF, NSST].

Circuit optimization is a very good example of difficulties in such integration. Despite considerable research activity in optimization of electronic circuits, optimization techniques have not been used as widely as might be expected [BCSV, ChVl, NPSS, NSST]. The reasons which are most frequently named to explain this situation are: the difficulty of linking optimization packages with simulation programs, the inadequacy of the optimization algorithms used, and the sophistication of these tools.

The most efficient optimization techniques assume that the objective (and constraint) functions are differentiable, and their (at least) first derivatives are available [BKZ, BCSV, GMSW, HaMa, Hieb, MGH, Pow, Sch]. In the case of circuit optimization, when the objective (and constraint) functions are evaluated from circuit responses, quite often partial derivatives are obtained by approximation techniques, and then several software tools (e.g., numerical approximation of gradients, scaling and descaling routines, etc.) must be used for interfacing (general) optimization algorithms with circuit simulators.

An efficient organization of circuit optimization, with repeated circuit simulations, cannot be provided by traditional simulation programs which are usually batch-oriented, and which require a new, independent run for each modification of the circuit. A more flexible structure of simulators is needed in which different analyses can be performed selectively, "on demand", and in which it is possible to modify circuit elements during a simulation session. The simulators should have an "open" structure of a collection (or a package) of procedures rather than a "closed" program with one, fixed sequence of operations. SPICE-PAC [Zub1] is such a simulation package, derived from the popular SPICE-2G.6 circuit simulator [NPSS, Pede, Vlad]. SPICE-PAC, similarly to the SPICE-2G.6 program, does not provide the values of derivatives of evaluated circuit responses. In all applications which require gradient information (e.g., circuit optimization), the values of derivatives of circuit responses must be approximated by numerical methods. Then, however, as the number of software packages grows, systematic approach to communication between packages becomes more and more important, as it can significantly increase overall reliability and reduce the total development time.

There are two basic methods of communication between software packages, the so called reverse and indirect communication [GMPW, More]. Packages with reverse communication interact on a "master–slave" basis, which means that there always is one "controlling" (master) package and a number of "controlled" (slave) packages that are invoked to perform specific (minor) tasks, and which return to the "master" immediately after completion of the task. It is this controlling package that is responsible for coordination of all packages and interfaces; the control is thus "centralized" and for complex software systems it can become quite difficult to manage and maintain.

In the case of indirect communication, the components are organized in a hierarchical structure, and the overall coordination is rather simple; the top level is responsible for major decisions only; all detailed decisions and actions are "delegated" to lower-level components which follow the same principle but on a smaller scale. The overall coordination and control is thus "decomposed" into a number of (hierarchical) levels, with well-defined interactions between levels, and "information hiding" due to interactions betweens adjacent levels only. On the other hand, however, indirect communication offers much less flexibility than reverse communication scheme as all components must conform to fixed schemes of interaction and communication.

Reverse and indirect communication are briefly characterized in section 2 and then illustrated with two practical examples, circuit optimization and mixed-mode simulation. For both examples, circuit simulation is provided by SPICE-PAC, a simulation package derived from the popular SPICE simulator. Section 3 outlines basic features of the simulation package SPICE-PAC, describes its general structure and interaction with other packages. Section 4 discusses circuit optimization that uses SPICE-PAC for circuit analyses, i.e., for evaluation of the optimization objective and constraint functions. Mixed-mode simulation or simulation of mixed analog–digital circuits is presented in section 5. Section 6 concludes the paper.

2. REVERSE AND INDIRECT COMMUNICATION

Reverse communication is usually implemented in a star-like structure, around the central coordinating module, as shown in Fig.1; the "central coordinator" supervises execution of tasks, it selects the order of task activations, and it passes all relevant information between interacting tasks performing all required conversions and transformations of data. For more complex applications, with large number of tasks executed according to irregular patterns (which may depend upon the data), reliable implementation of the coordinator is a nontrivial problem requiring an expert knowledge of all tasks involved, their constraints and side-effects.

In the case of circuit optimization, "task-1" could represent an optimization module, "task-2" – numerical approximation of gradients, "task-3" – evaluation of objective and constraint functions for a particular optimization problem, and "task-4" – evaluation of circuit responses for a given set of parameters (determined by "task-1" or "task-2"). A possible sequence of interactions would be an invocation (from the central coordinator) of "task-1" to perform an optimization. Whenever

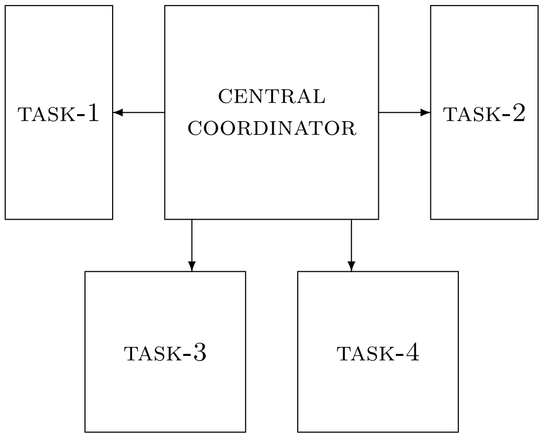

Fig.1. Global reverse communication.

"task-1" needs an evaluation of the objective and/or constraint functions or their derivatives, it "returns to the coordinator with a corresponding "request". Depending upon this request, the coordinator invokes "task-2" (for derivatives) or "task-3" (for objective and/or constraint functions). If numerical approximation of gradients ("task-2") needs a function value (e.g., for approximation based on perturbations), it "returns" to coordinator with a "request" that is directed to "task-3", etc.

In order to reduce complexity of central coordination, the set of tasks can be decomposed into a number of (relatively) independent parts with "local coordination" limited to subsets of tasks. This creates a simple form of hierarchical reverse communication, as shown in Fig.2. Further decomposition can be performed within subsets of tasks creating consecutive levels of hierarchical structure which reduces complexity of coordination but also restricts interactions between tasks to local subsets; all nonlocal interactions are usually coordinated by several control modules.

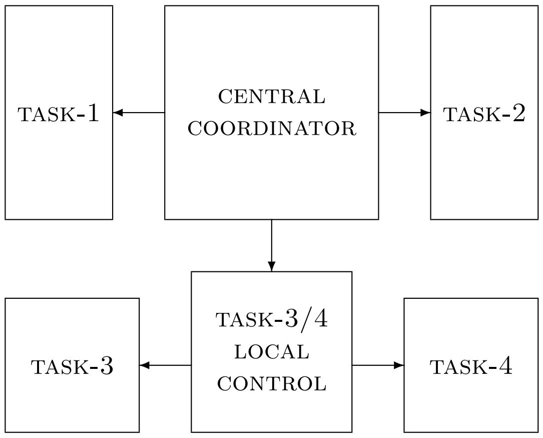

Fig.2. Hierarchical reverse communication.

Decomposition of central coordination into a hierarchical structure introduces a new type of interactions; the communication between the (remaining) central coordinator and local coordinators as well as among local coordinators is done at a "higher-level", i.e., a level that corresponds to operations of whole decomposed parts rather than individual tasks.

Systematic decomposition results in a hierarchical structure similar to one shown in Fig.3, which can be obtained from that shown in Fig.2 by one more decomposition of the central coordinator; it can be observed that in this "new" structure the central coordination has been practically eliminated.

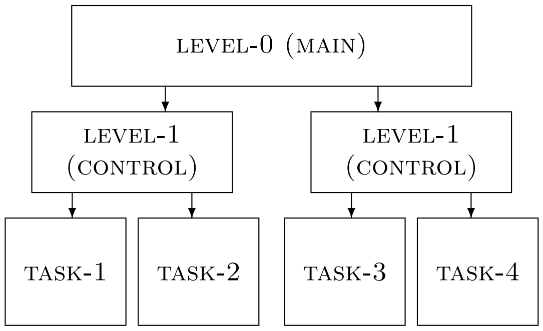

Fig.3. Hierarchical structure.

A significant advantage of hierarchical organization is abstraction that it offers at different level of hierarchy. For example, a transformation of the previous circuit optimization example into a hierarchical structure (shown in Fig.4) is slightly different as the hierarchy of tasks shown is rather a simple "path" than a binary structure of Fig.3; optimization module invokes the objective/constraint function and gradient evaluation ones, the gradient evaluation invokes the function evaluation, and finally the function evaluation calls the circuit simulator.

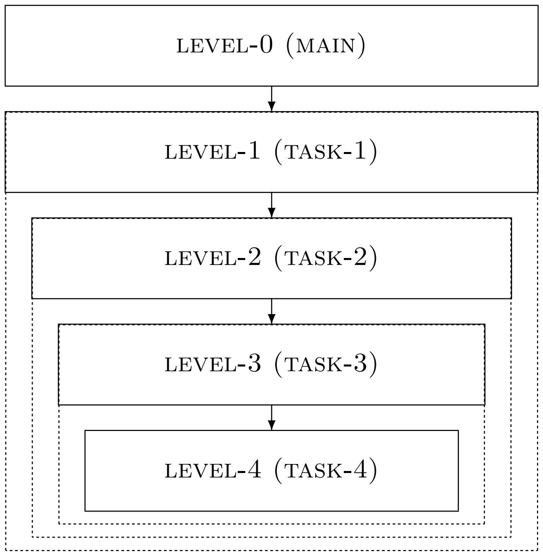

Fig.4. Hierarchical abstraction.

The levels of abstraction correspond to levels of hierarchical structure; at level-3, the evaluation function "includes" the circuit simulator (as indicated by broken lines), and any invocation of the function evaluation module "automatically" invokes the lower levels. Similarly, level-2 corresponds to evaluation of gradients evaluation with all the details (depending upon the evaluation method used) "hidden" at lower levels of the hierarchy. Level-1 is the "application level" which means that the whole hierarchy corresponds to a specific application determined by a particular composition of modules.

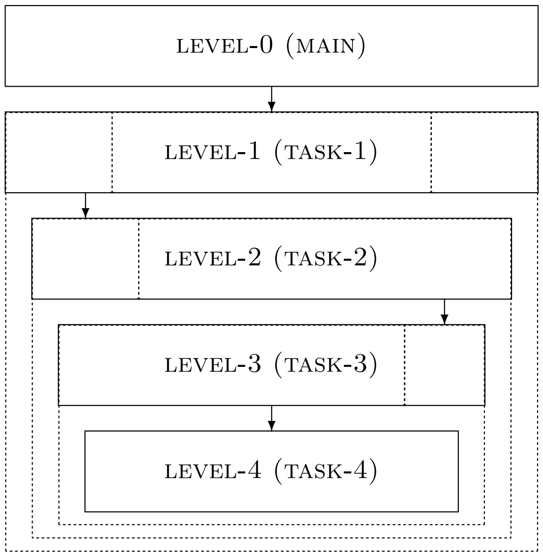

Fig.5. Indirect communication.

Quite often it is convenient to provide a number of functionally equivalent modules at the same levels of the hierarchical structure (e.g., different variants of the same method or different approaches to the solution of the same problem), as shown in Fig.5. Then, of course, a technique is needed for selection required variants at different levels of hierarchy and for passing relevant information to selected variants. Indirect communication is such a technique. Conceptually it corresponds to parameterized procedural abstraction with a number of parameters assigned to consecutive levels of the hierarchical structure (although in practical implementations this association may not be clear). In effect, a change of one or more parameters at the "top" level may result in a different behavior several levels "below".

In conclusion, for several interacting packages, reverse communication provides a "direct", flexible access to all packages, but it requires a detailed knowledge of all components to pass correct data structures and implement required sequencing. Indirect communication is less flexible than "loosely coupled" components of systems with reverse communication, but it offers better abstraction and structuring of packages. Indirect communication is thus a very convenient method for composition of packages; a gradient optimization package composed with a package for numerical approximation of derivatives results in a new package that provides optimization without derivatives, etc.

3. CIRCUIT SIMULATION

Computer-aided circuit analysis or circuit simulation, which matured in the 1970's, has established itself as a significant tool for analysis and design of integrated circuits. The SPICE-2 program [Pede, Vlad] developed at the University of California, Berkeley, has become one of the most popular "second-generation" circuit simulators. It provides several linear and nonlinear analyses, including DC operating point, nonlinear DC transfer curves, nonlinear transient, etc. Circuits may contain resistors, capacitors, inductors and mutual inductors, independent linear and nonlinear voltage and current sources, four types of dependent sources, transmission lines, and the four most common semiconductor devices: diodes, BJTs, JFETs and MOSFETs. SPICE has built-in models for semiconductor devices and the users need to specify only the pertinent model parameter values; moreover, if different semiconductor devices use the same model, the model parameters can be specified once only.

The SPICE program is, however, a "second generation" simulator which is batch-oriented and very inefficient for applications in which numerous variants of the same circuit need to be analyzed (e.g., circuit optimization). For such applications a new structure of the simulator is required, in which different analyses (for the same circuit topology) are performed "on demand", and in which there is an access to internal representation of circuit elements in order to update their values during optimization. The simulator should have an "open" structure of a collection (or a package) of procedures rather than a "closed" program with one, fixed sequence of operations. SPICE-PAC is such a simulation package, obtained by redesigning the SPICE 2G.6 simulation program.

SPICE-PAC is upwardly compatible with SPICE which means that it accepts the same input language (with only a few minor exceptions), and it provides the same set of circuit analyses, but it also supports a number of extensions which are not available in the original SPICE programs; a hierarchical naming scheme for (nested) subcircuits, static and dynamic circuit variables for modifications of circuit elements values during (interactive) simulation sessions, dynamic declarations of parameters and outputs, or parameterized subcircuit calls are examples of such extensions.

SPICE-PAC is organized into two levels; the "main" level contains 25 procedures (called SPICEA to SPICEY [Zub2]) that provide an interface with application-dependent programs, such as read a circuit description, update circuit variables, perform one of circuit analyses, etc. The second level collects all internal procedures ("invisible" to application programs) that implement operations of the main level. There is no restriction on the order or number of analyses performed within a single simulation session since all analyses are performed "on demand", by calling appropriate procedures of the package. Similarly, parameters of analyses, outputs or circuit variables can also be defined or updated any number of times by calling appropriate procedures. As shown in

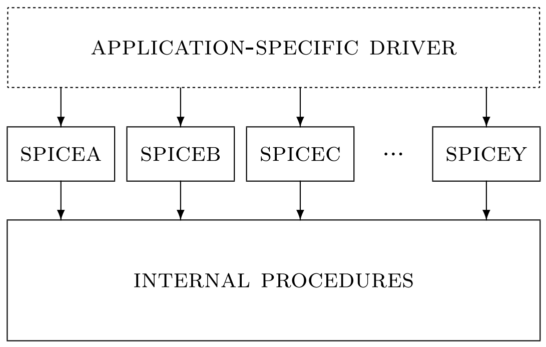

Fig.6. SPICE-PAC's organization.

Fig.6, it is the application-dependent part that controls the sequence of operations, as required by a particular application. Reverse communication is used between the application "driver" and the simulation package.

The SPICE-PAC project was oriented toward two basic applications, interactive simulation and circuit optimization. In interactive circuit simulation, the driving routine mainly handles communication with the user, i.e., it enters user commands, converts them into corresponding sequences of SPICE-PAC's interfacing procedure invocations, and displays (in numerical or graphical form) the results. The complexity of the driver is directly related to the sophistication of interaction. In typical applications the interaction is rather simple, and it can use, for example, commands similar to the SPICE input language. A convenient representation of results appears to be a more troublesome issue because of a variety of available output devices and presentation methods.

In circuit optimization, the driving routine has to control at least two packages, the optimization package and the circuit simulator. In many cases, when indirect communication is used for interfacing circuit simulation with optimization, the packages are hierarchically structured, and then the whole optimization process is controlled by the selected optimization algorithm. This also means that the user can influence the optimization process only by selection of the starting point and (some of) the optimization parameters. More flexible (but also more difficult) solution is to use reverse communication in which case user routines (or user - in an interactive way) can control the optimization process at the step level, and can adjust the parameters even during the optimization process.

4. CIRCUIT OPTIMIZATION

The most effective optimization methods assume that the objective (and constraint) functions are differentiable, and their (at least) first derivatives are available [BCSV, GMPW]. In some cases, however, even if the derivatives can be obtained analytically, the amount of calculations can be quite excessive, and then numerical approximation of gradients can be the simplest and the most effective solution [Broy]. This can be done directly by differencing function values, i.e., by evaluating differences of function values which correspond to small perturbations of independent variables (one at a time), or indirectly, by updating the initial approximation in consecutive iteration steps using for example Broyden formula [Broy] or one of its modifications [Mad].

Optimization algorithms are designed to solve classes of problems determined by properties of the objective and constraint functions. The most popular types of objectives [GMPW, HoSch, Sch] include linear objective function(s), quadratic objective function(s), sum of squares of linear functions, generalized polynomial objective function(s), general objective function(s), sum of squares of general functions. The popular types of constraints are unconstrained problems, bounded problems (constant constraints), linear constraint functions, sparse linear constraint functions, quadratic constraint functions, generalized polynomial constraint functions, general (nonlinear) constraint functions. Within the same class of objective and constraint functions, different algorithms may provide different rates of convergence (linear, superlinear, quadratic, etc.), they may use different interaction mechanisms (passing information through the parameter lists or global variables and structures), different data structures (static or dynamic arrays, uniform one dimensional arrays or multidimensional structures, etc.), and may have significantly different requirements for (user-supplied) working space. Yet another important consideration is the control structure that connects the definition of the objective and constraint functions and (possibly) the derivatives with the optimization algorithm (usually it is a user-defined procedure which can have a fixed or quite flexible form). The most popular solution [GMSW, HaMa, HoSch, Mad, More] is to use a fixed structure for the user-supplied procedure and to pass the name of this procedure as one of optimization parameters (which is a simple form of indirect communication). Some of the optimization codes use reverse communication for interfacing user definitions [GMPW, Pow, Sch].

The choice of an optimization algorithm depends not only upon the type of problems to be solved, but also upon the available information about performance of the algorithm, i.e., the number of function evaluations, the workspace size, the convergence properties, and the "behavior" of the algorithm during extensive testing on problems that are believed to be typical [GMPW, HoSch, Sch]. However, because proliferation of "good" algorithms proved to be an ineffective means for providing high quality software, an awareness developed of the need for program libraries. A program library is a set of routines that are conceived and written within a unified framework, to be available to a general community of users. A pilot version of such an optimization library has been derived from existing, modified, redesigned, restructured or simply new optimization codes.

This library is composed of three layers. The central layer contains the "core" routines which implement reliable, efficient and general optimization algorithms. The bottom layer extends the core by a number of auxiliary routines (e.g., scaling routines, approximation of gradients, etc.) which are identical (or almost identical) for many core routines. Both layers support reverse as well as indirect communication. The top layer is composed of simple drivers (or service routines [GMPW]) which use indirect communication facilities of remaining levels to compose several routines into independent (and simple-to-use) packages. Presently, the library contains two core optimization algorithms, an implementation of minimax optimization with linear constraints proposed by Hald [HaMa] and later modified and extended several times, and an implementation of a very reliable Han-Powell method for minimization with general constraints [Han, Pow] (the method is quite attractive for circuit optimizations because of extremely low number of function evaluations [Sch], however, its workspace requirements are rather high). Since both algorithms require partial derivatives of all functions (objective as well as constraint) with respect to all optimization variables, routines for numerical approximation of gradients have been included in the library. A composition of SPICE-PAC with this pilot optimization library is shown in Fig.7. Indirect communication is used for accessing objective and constraint evaluation routines, and reverse communication for circuit simulation operations.

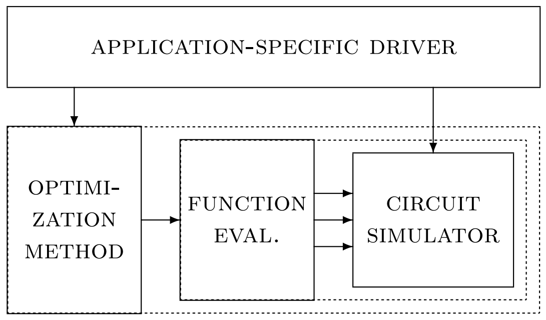

Fig.7. Organization of circuit optimization.

As an example of circuit optimization, a single stage CE amplifier in a self-biasing configuration is analyzed, and it is to find the values of resistors R1, R2 and RE, such that for the midband frequency f=50KHz, and for the temperatures between -50 and 100 degrees Celsius, the voltage gain is equal (or, as close as possible, rather) to 10V/V, and the input resistance is not less than 10Kohms. The minimax optimization is used in this example.

```
** INPUT LISTING -- TEMPERATURE = 27.0 DEG C
** single stage CE amplifier optimization 1
VCC 5 0 12
VIN 1 0 AC 1
R1 2 5 100K
R2 2 0 10K
RC 4 5 5K
RE 3 0 300
CB 1 2 100UF
Q1 4 2 3 MOD
.MODEL MOD NPN(BF=50 VAF=50 IS=1.E-9 RB=100)
```

```
.PRINT AC V(4) V(2) I(VIN)
.AC 50K
.END/EXT
.VAR R1
.VAR R2
.VAR RE
.PAR/1 TEMP(-50.0)
.PAR/2 TEMP(27.0)
.PAR/3 TEMP(100.0)
.END
```

In minimax formulation, the optimization variables are R1, R2 and RE, and the $2k+1$ residual functions (k is the number of discrete temperature values; in this example k=3) are:

- the differences between the magnitude of the voltage gain and 10 V/V for the selected temperatures (k residual functions),
- the differences between 10 V/V and the magnitude of the voltage gain for the selected temperatures (k residual functions),
- the difference between 10K and the minimum input resistance (decreased 100 times).

The trace of optimization steps is as follows (`max` denotes the maximum residual function, i.e., the maximum absolute value of the difference between 10V/V and the voltage gain):

	R1	R2	RE	max
1	1.00d+05	1.00d+04	3.00d+02	6.91d+01
2	1.21d+05	9.58d+03	3.98d+02	4.62d+01
3	1.96d+05	1.49d+04	4.59d+02	2.24d+01
4	2.86d+05	2.32d+04	4.74d+02	6.01d-01
5	4.01d+05	2.89d+04	4.25d+02	5.14d-01
6	4.47d+05	3.54d+04	4.45d+02	2.69d-01
7	2.65d+05	1.89d+04	4.54d+02	1.02d+01
8	4.03d+05	3.08d+04	4.39d+02	2.20d-01
9	3.55d+05	3.25d+04	4.51d+02	7.35d-02
10	3.45d+05	3.34d+04	4.52d+02	2.17d-02
11	3.39d+05	3.38d+04	4.53d+02	4.63d-03
12	3.38d+05	3.38d+04	4.53d+02	3.25d-03
13	3.38d+05	3.38d+04	4.53d+02	3.36d-03

After 13 iteration steps the maximum residual function is less than 0.004. In many cases a less accurate solution, obtained in a few iteration steps, should be satisfactory.

5. MIXED–MODE SIMULATION

As the analog and digital process technologies merge, so must the corresponding simulation tools. The phrase "mixed-mode" simulation has been used to refer to the simulation of electrical networks consisting of both analog and digital parts, regardless of the level of design abstraction.

Mixed-mode simulation has been gaining popularity because of analog circuitry that exists in virtually all digital systems. It appears, however, that even among sophisticated users of design automation tools, the analog and digital portions of a system are usually designed and simulated separately. It also appears, that it is rather difficult to correlate these two distinct simulations in order to analyze how analog and digital parts affect each other. Clearly, an "integrated" approach is needed which can handle both analog and digital simulation within one, consistent simulation environment [Goer]. A number of modifications and extensions have been implemented in SPICE-PAC to allow simulation of circuits with digital elements described at the gate or block level. General organization of mixed-mode simulation is shown in Fig.8.

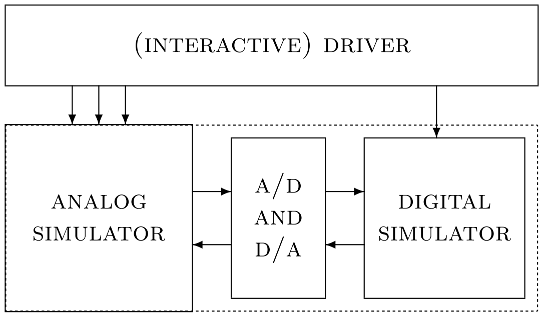

Fig.8. Mixed-mode simulation.

Any implementation of integrated mixed-mode simulation must solve two basic questions, (i) conversion of analog to digital and digital to analog information on interfaces of analog and digital components, and (ii) synchronization of the (usually variable) timesteps of the analog simulation [VlSi] with the event list that drives the (event-driven) digital simulation [MME]. The analog-to-digital conversion can be handled by establishing voltage thresholds and corresponding digital signals (the conversion is performed by elements called "thresholders"). The digital-to-analog conversion is much more difficult because it must generate a continuous analog signal on the basis of discrete digital values. Two popular and simple solutions assume that the converted waveforms are (1) piecewise linear (with averaged levels) or (2) piecewise exponential.

A basic analog-digital interface implemented in SPICE-PAC provides 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 parts; 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 [Zub1] 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
```

In the following simple example, the digital part (not shown here) is a two-input one-output block, with inputs indicated by the PUTLIST and the output by GETLIST. It should be observed that there is one common conversion table for analog-to-digital and digital-to-analog conversions, and that the rise and fall rates are different:

```
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:TCONV V(2),V(3)
.GETLIST:TCONV:TDEL VX
.TABLE TCONV (-5.0,-1,0.0,1,+5.0)
.TABLE TDEL (2E-7,5E-8,2E-8)
.END
```

The implementation of time-domain (transient) analysis has been modified in such a way that if both PUTLIST and GETLIST are nonempty the analog-digital interfacing routines are invoked for each successfully solved timepoint [VlSi]. The interfacing routines perform analog-to-digital conversion of all PUTLIST voltages, and then check if any digital value created during this conversion differs from the corresponding (digital) value created at the previous timepoint. If there is at least one "new" digital value after the conversion, the timepoint is (iteratively) adjusted to a value corresponding to the closest conversion threshold, and then the interfacing routine is invoked to perform a single step of digital simulation (at the gate, functional or behavioral level). After completion of this step, 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 simulation, and the simulation resumes.

The gate-level description of the digital part is composed of three parts, "input", "output" and "circuit"; for example, a description of a two-input one-output block for the analog part shown earlier

```
input : x1,x2;
output : y;
circuit
   y:=xor(x1,x2)
end
```

where the "input" signals x1 and x2 correspond to consecutive (analog) PUTLIST elements, and the "output" signal y to the GETLIST voltage source.

The trace of all changes of digital signals is as follows:

```
time :  0.0000d+00  inp  -1 -1  out  -1
time :  6.4360d-07  inp  -1  1  out   1
time :  1.1986d-06  inp   1  1  out  -1
time :  2.6536d-06  inp   1 -1  out   1
time :  3.0649d-06  inp  -1 -1  out  -1
time :  5.6436d-06  inp  -1  1  out   1
time :  6.1541d-06  inp   1  1  out  -1
time :  7.6536d-06  inp   1 -1  out   1
time :  8.0716d-06  inp  -1 -1  out  -1
```

It can be observed that the "delayed" events are the only events that are analyzed without changes of (digital) input signals.

6. CONCLUDING REMARKS

Two different mechanisms for implementation of software interfaces have been discussed and illustrated with simple examples of circuit optimization and mixed-mode simulation. Indirect communication provides a simple technique for hierarchical structuring and tuning integrated tools to particular needs. Reverse communication seems to be "invaluable" mechanisms in new or unconventional applications as it provides unrestricted access to all components of an integrated system (it may, however, be very "troublesome" for inexperienced users). It is anticipated that a combination of both communication techniques will prove useful in practical applications.

In some cases, considerable improvements can be obtained by restructuring and redesigning existing programs. A conversion of the SPICE circuit simulation program into a simulation package significantly reduces difficulties and inefficiencies associated with interfacing circuit simulation with other software tools.

Systematic approach to design and organization of computer-aided design software is needed. A good example has been set by "mathematical software" or software which implements wodely applicable mathematical procedures; NAG, IMSL and NATS are examples of successful projects that resulted in widely used collections of high-quality numerical software [Cowe]. Computer-aided design software is more difficult to unify and "standardize" as it often lacks the rigor of numerical algorithms, however, if its growing popularity is the best indication that it should follow the best examples of mathematical libraries.

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.

References

[AnHu] K.J. Antreich, S.A. Huss, "An interactive optimization technique for the nominal design of integrated circuits"; IEEE Trans. Circuits and Systems, vol.31, no.2, pp.203-211, 1984.

[BKZ] J.W. Bandler, W. Kellermann, W.M. Zuberek, "A comparison of recently implemented optimization techniques"; Proc. European Conf. on Circuit Theory and Design, Stuttgart (ECCTD'83), West Germany, 1983.

[BCSV] 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.

[Broy] C.G. Broyden, "A class of methods for solving nonlinear simultaneous equations"; Mathematics of Computation, vol.19, pp.577-593, 1965.

[ChVl] E. Christen, J. Vlach, "NETOPT - a program for multiobjective design of linear networks"; IEEE Trans. on Computer-Aided Design, vol.7, no.5, pp.567-577, 1988.

[Cowe] W.R. Cowell (ed.), "Sources and development of mathematical software"; Prentice-Hall 1984.

[FiNi] J.K. Fidler, C. Nightingale, "Computer Aided Circuit Design"; Wiley & Sons 1978.

[GMPW] P.E. Gill, W. Murray, S.M. Picken, M.H. Wright, "The design and structure of a Fortran program library for optimization"; ACM Trans. Mathematical Software, vol.5, no.3, pp.259-283, 1979.

[GMSW] P.E. Gill, W. Murray, M.A. Saunders, M.H. Wright, "Procedures for optimization problems with a mixture of bounds and general linear constraints"; ACM Trans. Mathematical Software, vol.10, no.3, pp.282-298, 1984.

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

[HaMa] J. Hald, K. Madsen "Combined LP and quasi-Newton methods for minimax optimization"; Mathematical Programming, vol.20, no.1, pp.49-62, 1981.

[Han] S.P. Han, "Superlinearly convergent variable metric algorithms for general nonlinear programming problem"; Mathematical Programming, vol.11, pp.263-282, 1976.

[Hieb] K.L. Hiebert, "An evaluation of mathematical software that solves systems of nonlinear equations"; ACM Trans. Mathematical Software, vol.8, no.1, pp.5-20, 1982.

[HoSch] W. Hock, K. Schittkowski, "Test examples for nonlinear programming codes" (Lecture Notes in Economics and Mathematical Systems 187); Springer Verlag 1981.

[LKCF] J.C. Lai, J.S. Kueng, H. Chen, F. Fernandez, "ADOPT - a CAD system for analog circuit design"; Proc. IEEE 1988 Custom Integrated Circuits Conf., pp.3.2.1-3.2.4, 1988.

[Mad] K. Madsen, "Minimax solution of nonlinear equations without calculating derivatives"; Mathematical Programming Study 3, pp.110-126, 1975.

[More] J.J. More, "Notes on optimization software"; NATO Advanced Research Institute on Nonlinear Optimization, Cambridge, England, 1981.

[MGH] J.J. More, B.S. Garbow, K.E. Hillstrom, "Testing unconstrained optimization software"; ACM Trans. Mathematical Software, vol.7, no.1, pp.17-41, 1981.

[MME] S. Murai, H. Matsushita, K. Enomoto, "Logis simulation programs"; in: "Logic design and simulation", E. Hoerbst (ed.), pp.135-163, North-Holland 1996.

[NPSS] 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.

[NRST] W. Nye, D.C. Riley, A. Sangiovanni-Vincentelli, A.L. Tits, "DELIGHT.SPICE : an optimization-based system for the design of integrated circuits"; IEEE Trans. on Computer-Aided Design, vol.7, no.4, pp.501-519, 1988.

[NSST] B. Nye, A. Sangiovanni-Vincentelli, J. Spoto, A. Tits, "DELIGHT.SPICE, an optimization-based system for the design of integrated circuits"; Proc. 1983 Custom Integrated Circuits Conf., Rochester NY, 1983.

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

[Pow] M.J.D. Powell, "A fast algorithm for nonlinearly constrained optimization calculations"; in: "Numerical Analysis" (Lecture Notes in Mathematics 630), G.A. Watson (ed.), pp.144-157, Springer Verlag 1978.

[Sch] K. Schittkowski, "Nonlinear programming codes - information, tests, performance" (Lecture Notes in Economic and Mathematical Systems 183), Springer Verlag 1980.

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

[Vlad] [21] 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.

[Zub1] W.M. Zuberek, "SPICE-PAC 2G6c - An Overview"; Technical Report #8903, Department of Computer Science, Memorial University of Newfoundland, St. John's, Newfoundland, Canada A1C 5S7, 1989.

[Zub2] W.M. Zuberek, "SPICE-PAC 2G6c - User's guide"; Technical Report #8902, Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada A1C-5S7, 1989.