THE DESIGN OF AN HIERARCHICAL CIRCUIT-LEVEL SIMULATOR

M Zwolinski and K G Nichols

Department of Electronics, University of Southampton, United Kingdom

1. INTRODUCTION

The increasing size and complexity of VLSI circuits developed in the past few years and of those that will be developed in the forseeable future, means that existing circuit-level simulators, designed for the analysis of much smaller circuits, are no longer adequate (e.g. SPICE [1], ASTAP [2] etc.). This problem cannot be overcome by simply running these simulators on larger and faster computers. The simulator described in this paper, although still under development, has been designed to overcome the limitations imposed by the data structures and algorithms of such simulators.

One of the major drawbacks of existing simulator programs is that they have traditionally been written in FORTRAN. While FORTRAN is undoubtedly an excellent language for purely numerical operations, it lacks many of the subtleties present in more modern block-structured languages such as Pascal. For example, the only data structures allowed in FORTRAN are arrays, whereas dynamically allocatable data types are available in e.g. Pascal. Thus, flexibe tree and list structures may be defined, which, for example, allows a network Jacobian matrix to be set up and solved without the constraints and inefficiencies imposed by array bounds. Furthermore, FORTRAN, unlike modern block-structured languages does not permit recursion, which, as will be seen, is of particular importance in this simulator. Moreover, we have tried to apply the concept of functional programming, as described by Henderson [3], to our design. Briefly, this means that subprograms return one result from a number of arguments without altering the values of global variables. This ideal can produce large inefficiencies that are unacceptable in a circuit simulator, so some compromises have had to be made.

Our simulator is specifically designed for the analysis of hierarchically partitioned circuits. The reasons for partitioning a circuit and the theoretical justification for so doing will be explained in Section 2. Section 3 shows how the features of a block-structured language are applied to the analysis of a partitioned circuit. Section 4 considers the use of individual timesteps for each subcircuit, how the concept of latency can be applied and how the analysis of the circuit with respect to time can be controlled. Lastly, section 5 extends the concept of an hierarchically partitioned circuit to include mixed-mode analysis.

2. CIRCUIT PARTIONING THEORY

Why Partition a Circuit?

Partitioning a circuit into a number of hierarchically nested subcircuits will not, in itself, reduce the computational effort needed to analyse the circuit. This is especially true in competition with conventional simulators using sparse matrix techniques. In a VLSI circuit, it is likely that at any given time, a few node voltages will be changing relatively quickly while the majority of node voltages remain fairly constant. By partitioning a circuit, the more active nodes will be confined to a few subcircuits. Thus, if each subcircuit is allowed to have its own timestep, it can be analysed only as often as it needs to be. The analysis of such a circuit is considered further in Section 4. Further savings in computation can be made if various subcircuits are analysed at a logic level, as will be discussed in Section 5.

From the point of view of a VLSI circuit designer, partitioning a circuit means that a repeated structure need only be described once. However, the partitioning must be done manually, although this should not be a serious problem, since the design of a circuit would normally proceed in a top-down manner, in which functional blocks are continually refined.

Formulation and Solution of Partitioned Circuit Equations

The equations for a general nonlinear circuit can be expressed as:

$$f(x, \dot{y}, t) = 0 \qquad (1)$$

where y = y(x) and x is a vector of circuit variables (voltages and currents). After discretisation in time, these equations are reduced to the form:

$$g(x_n) = 0 \qquad (2)$$

where $x_n = x(t_n)$.

Equation (2) is solved numerically using the Newton-Raphson formula, i.e. by solving:

$$J_g^m x_n^{m+1} = J_g^m x_n^m - g(x_n^m) \qquad (3)$$

where J_g^m is the Jacobian at the m-th iteration at t_n. Equation (3) is equivalent to

$$J^m x^{m+1} = F^m \qquad (4)$$

where F is the vector of excitations. In practice, J and F are built with the aid of companion models, e.g. Calahan, Chapter 3 [4].

The network equations, equivalent to equation (4), for one subcircuit are shown in equation (5), which is an extension of the torn node technique of Linardis and Nichols [5]. Equation (5) is formulated using the modified nodal analysis of Ho et al [6], thus allowing branch currents to be included as circuit variables whereas the equivalent equation in [5] is formulated with nodal analysis.

$$\begin{pmatrix} Y_{11}^m & A_{12}^m \\ A_{21}^m & A_{22}^m \end{pmatrix} \begin{pmatrix} v^o \\ x^{m+1} \end{pmatrix} = \begin{pmatrix} I^m \\ F^m \end{pmatrix} + \begin{pmatrix} i^o \\ 0 \end{pmatrix} \qquad (5)$$

where v is the vector of node voltages at the external (torn) nodes of the subcircuit; i is the vector of currents flowing into the torn nodes from the rest of the circuit; Y_{11}, A_{12}, A_{21} and A_{22} form the Jacobian matrix of the subcircuit and I and F are excitation vectors.

By means of Gausian elimination, the equations of a subcircuit as given by equation (5) may be transformed to equation (6) (in which the iteration superscripts are omitted for clarity)

$$(Y_{11}-A_{12}A_{22}^{-1}A_{21})\,v=(I-A_{12}A_{22}^{-1}F)+i \quad (6)$$

which represents a number of Norton circuits (or macromodels) which are equivalent to the external characteristics of the subcircuit and in which the internal circuit variables, x, have been supressed.

Equation (6) may now be used to embed a subcircuit macromodel into the equations of another subcircuit and the torn node currents, i, for all the subcircuits embedded within one other sum to zero. This leads to a recursive linearisation and solution algorithm for the entire circuit at a time t .

Starting at the deepest level of nesting, i.e. those subcircuits that have no further subcircuits embedded within them, each subcircuit's equations are linearised using equation (7).

$$A_{22}^{m}x^{m+1}=F^{m}-A_{21}^{m}v^{o} \quad (7)$$

The torn node voltages, v , are assumed to be correct during this inner iteration loop. When sufficient convergence has been achieved, each subcircuit's equations are reduced to the form of equation (6) and included in the equations of the embedding subcircuit.

This linearisation, reduction and substitution is repeated for the equations of each subcircuit until a linear set of equations is obtained for the main circuit (i.e. that subcircuit which is not embedded in any other). Beginning with the main circuit, the internal variables, x, are calculated from equation (7) and recursively back-substituted into the equations of each subcircuit as torn node voltages, v. Hence the circuit variables of every subcircuit are calculated.

The entire algorithm is now repeated until convergence is reached for every variable, x, of every subcircuit.

At each iteration of this outer loop, new values of v for each subcircuit are calculated. Therefore, it is not essential that absolute convergence is obtained in the linearisation of each subcircuit. Indeed, future research may show that one iteration loop is sufficient for each set of subcircuit equations.

3. IMPLEMENTATION OF AN HIERARCHICAL CIRCUIT-LEVEL SIMULATOR IN A BLOCK STRUCTURED LANGUAGE.

Recursion

In order to exploit fully the concept of a circuit consisting entirely of nested subcircuits, each subcircuit should be treated in exactly the same way as any other. The action of the simulator should not depend upon a subcircuit's position in the structure of the circuit. It is possible to implement an analysis program satisfying this criterion as an iterative algorithm, but a better solution is the use of a recursive algorithm. Thus the analysis procedure is invoked by itself once for each subcircuit.

Data structure of an Hierarchically Partitioned Circuit

The subcircuit data structure for a recursive simulator algorithm should contain pointers to the subcircuit's immediate neighbours but should not contain any reference to the subcircuit's position in the overall circuit structure.

Each subcircuit is embedded within exactly one other subcircuit. Therefore, the subcircuit structure has one pointer to the embedding subcircuit. There is, however, no upper or lower limit to the number of subcircuits that may be nested within a given subcircuit. In order to avoid imposing limits and to avoid wasting storage, a list of nested subcircuits is used in preference to an array of pointers. Each member of the list may have a list of embedded subcircuits which leads to a binary tree structure.

A simple example of the tree structure is shown in figure (1).

A recursive Pascal-like procedure to move through such a data structure is shown below, where the 'child' of a subcircuit is the first member in a list of embedded subcircuits, and the 'brother' of a subcircuit is the next subcircuit in the same list as the current subcircuit.

```
procedure search (subcircuit pointer);

begin

if subcircuitpointer ^. child < > nil then

      search (subcircuitpointer ^. child);

if subcircuitpointer ^. brother < > nil then

      search (subcircuitpointer ^. brother)

end ;
```

The procedure would initially be invoked by:-

```
search (maincircuit);
```

This depth-first search procedure is the outline of the forward elimination and back-substitution phases described in Section 2.

Circuit Analysis

Data Structures and Algorithms

In addition to the data structure described for nesting subcircuits, further structures are required for the internal representation of subcircuits. It has been decided to store circuit elements as linearised companion models rather than as complex devices. This transformation is performed by a preprocessing program. The preprocessor accepts a free-format circuit description, which allows comments to be mixed in with keywords and data. A certain amount of circuit integrity checking is done and the nodes of each subcircuit are renumbered so as to reduce the number of infills, i.e. zero locations in the subcircuit Jacobian, which become non-zero when the network equations are solved, [6].

The preprocessor then writes the new version of the circuit to an intermediate file.

The major advantage of this approach, in which network elements are stored as companion models is to reduce the dependence of the simulator upon device models. Hence, new models may be added with comparative ease. To calculate the instantaneous values of companion models, however, each model must include symbolic information as to what the model represents. Furthermore, each model needs a list of parameters from which its instantaneous value is calculated. Thus, the dependence of the simulator upon device types is limited to one group of procedures, in which the symbols identifying each model are interpreted. The parameters associated with each model are substituted for the symbols in order to evaluate the companion model. To add a new model to the simulator therefore requires the inclusion of extra interpretation code.

It was noticed that the evaluation of companion model values from user defined variables and node voltages resulted in the repetition of a number of operations within each Newton-Raphson iteration loop. This is especially true for complex nonlinear devices such as MOSFETS. It is possible to avoid this repetition by setting a number of program variables within each iteration loop. This method, however, conflicts with the principle of functional programming. Nevertheless, an acceptable compromise has been reached by including in the parameter lists of nonlinear models, a number of parameters that are shared between models and recalculated once in each Newton-Raphson iteration. Hence the repetition of identical operations is avoided.

Once the values of a subcircuit's companion models have been calculated, the network matrix is set up and solved as part of each Newton-Raphson iteration. The sparse matrices of each subcircuit are stored as lists of rows, which are themselves lists. Each companion model value may be included in the matrix at up to four different locations. Therefore, in order to avoid searching for the matrix locations at which each companion model is to be added, we have included pointers in the data structure, so that when a model value is calculated, it may be added into the sparse matrix directly.

To reduce the amount of computation performed in the course of the simulation, the sparse matrix structure is set up once at the beginning of the simulation. In addition, a symbolic solution of the sparse matrix is performed so that locations for infills are included in each matrix. Consequently, the data structure used to represent a VLSI circuit is potentially very large. It is assumed that the simulator will be run on a computer with a virtual memory

4. TIMESTEPS AND LATENCY IN SUBCIRCUITS

The timestep used for the analysis of a circuit is determined by the fastest changing node voltage. In general, the faster a voltage changes, the smaller the timestep must be to ensure convergence in a Newton-Raphson iteration. In a large unpartitioned circuit, it is unlikely that every node voltage will be changing at the same rate. Indeed, it is probable that a large number of node voltages will not change between time points. Consequently, the voltages of many nodes in such a circuit will be calculated more often than they need be.

If a circuit is partitioned into small hierarchically nested subcircuits, however, each subcircuit may be analysed with a timestep determined by the subcircuit's most active node rather than by the most active node in the entire circuit. Thus each subcircuit is only analysed when one or more of its node voltages has changed significantly since the last analysis time point. If the macromodel of a subcircuit is required for embedding in another subcircuit's equations before the macromodel is due to be recalculated, then the macromodel is assumed to be unchanged from the last analysis time point. This assumption is generally known as latency. At any given time point in the analysis, between one and all of the subcircuits in a circuit may have to be analysed. Consequently some form of overall control is needed to determine the next analysis time point and which subcircuits are to be analysed at that timepoint. The control structure takes the form of an event schedular. Event schedulars have been used for many years in logic simulators and we use an event schedular similar to that used in SPLICE [7] for mixed-mode analysis. Our event schedular consists of two lists. The first list is of those subcircuits due to be analysed at the present timepoint and the second list is of timepoints in the future together with the subcircuits due to be analysed at those timepoints. The timestep of each subcircuit is controlled by the local truncation error, Shichman, [8]. Therefore, it is possible to predict the next timestep for each subcircuit and hence to update the event lists after each analysis time point.

5. MIXED-MODE ANALYSIS

Since subcircuits are analysed in isolation, it is possible to characterise some subcircuits at a logic level rather than at a circuit level. It has been decided that logic elements and circuit elements maynot be mixed within subcircuits, unlike DIANA [9]. Therefore, logic to circuit and circuit to logic interfaces are confined to subcircuit boundaries. This stipulation simplifies the simulator program and does not unduly constrain circuit designers.

The simulation of a subcircuit at a logic level increases the speed of the analysis but at the expense of accuracy. It is likely, however, that subcircuits would only be analysed at a logic level once their design had been verified at a circuit level, although the characterisation of a logic subcircuit from a circuit level description will probably have to be done manually.

Bearing in mind the simplicity of logic elements, it is probable that the interfaces between circuit level subcircuits and logic level subcircuits will simply consist of the mapping of a logical value directly onto a voltage and vice versa. Greater accuracy may be obtained by modelling the outputs of different families of logic elements as Boolean controlled Thevenin circuits as described by Arnout and de Man [9]. Alternatively logic level subcircuits may be buffered to circuit level subcircuits by characterising external gates at circuit level.

6. CONCLUSION

This paper has described the design philosophies behind a simulator for VLSI circuits. Many details are necessarily incomplete because the simulator is still under development. We have sought, however, to illustrate how these design principles have been applied to the representation and solution of hierarchically partitioned circuits.

REFERENCES

1. Nagel, L., 1975, "SPICE2: A Computer Program to Simulate Semiconductor Circuits", University of California, Berkeley, U.S.A., Electronics Research Laboratory Report, No. ERL-M520.

2. Weeks, W. T., Jimenez, A. J., Mahoney, G. W., Mehta, D., Qassemzadeh, M. and Scott, T. R., 1973, IEEE Trans. Circuit Theory, C T-20, 628-634.

3. Henderson, P., 1980, "Functional Programming", Prentice-Hall International, London, U.K.

4. Calahan, D. A., 1972, "Computer-Aided Network Design", Revised Ed., McGraw-Hill, New York, U.S.A.

5. Linardis, P. and Nichols, K. G., 1979, "Partitioning with Latency Exploitation in the Time-Domain Analysis of Large Nonlinear Electronic Circuits", Proc. IEE CADMECCS, Brighton, U.K.

6. Ho, C.-W., Ruehli, A. E. and Brennan, P. A., 1975, IEEE Trans Circuits and System, CAS-22, 504-509.

7. Newton, A. R. and Pederson, D. O., 1978, "A Simulation Program with Large-Scale Integrated Circuit Emphasis", Proc. IEEE ISCAS, New York, U.S.A.

8. Shichman, H., 1970, IEEE Trans. Circuit Theory, CT-17, 378-386.

9. Arnout, G. and DeMan, H. J., 1978, IEEE Journal Solid-State Circuits, SC-13, 326-332.

One of the authors MZ would like to acknowledge the support provided by the SERC and IBM United Kingdom Laboratories Limited, Hursley Park, Winchester.

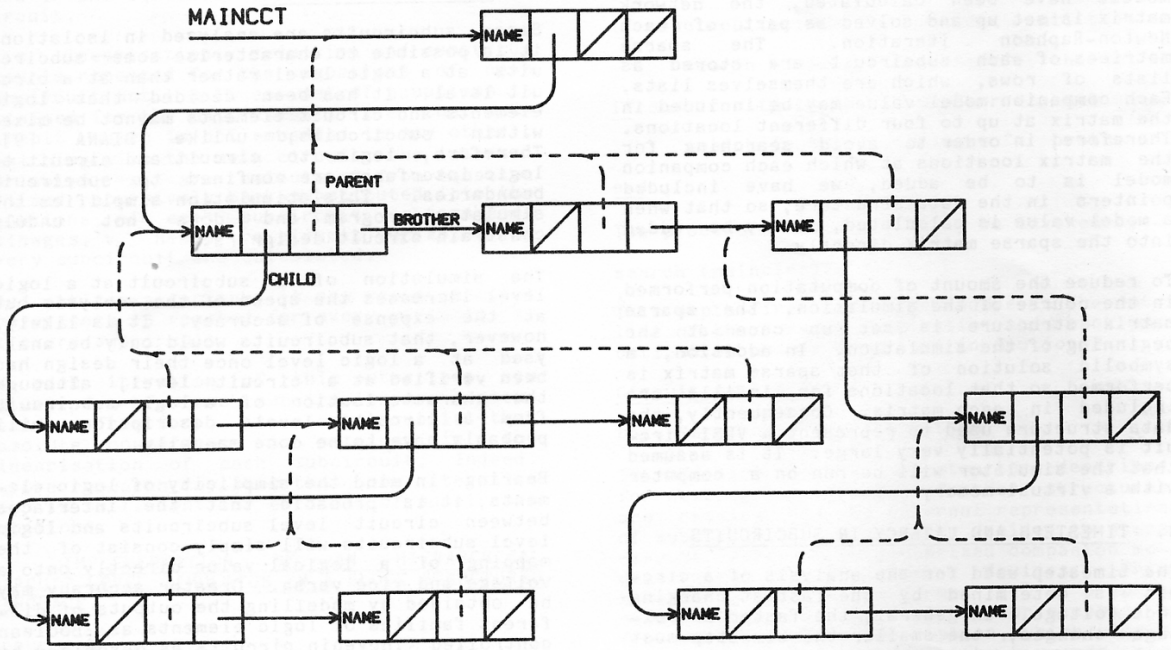

Figure 1. Example of Data Structure of a Partitioned Circuit