SAMSON2: An Event Driven VLSI Circuit Simulator

KAREM A. SAKALLAH, MEMBER, IEEE, AND STEPHEN W. DIRECTOR, FELLOW, IEEE

Abstract—**This paper describes the circuit simulation algorithms of the SAMSON2 mixed circuit/logic simulator. SAMSON2 employs event driven simulation techniques to exploit the inherent temporal sparseness in VLSI circuits. At the circuit level, SAMSON2 can be more than an order of magnitude faster than a "standard" circuit simulator such as SPICE2 while providing comparable waveform accuracy. This performance is achieved by partitioning a circuit into subcircuits which are allowed to have separate integration step sizes reflecting their activity level. The accuracy of the solution is maintained by controlling the errors due to decoupling among the subnetworks as well as the local truncation within each subnetwork. Simulation examples of typical MOS integrated circuits (IC's) are presented.**

I. INTRODUCTION

THE simulation of integrated circuit (IC) chips at the detailed electrical level continues to be an indispensable element in the overall IC design and verification process. However, the dramatic increase in the complexity of IC's over the past decade has strained the capabilities of traditional circuit simulators such as SPICE2 [13] and ASTAP [27]. For circuits containing more than a few hundred transistors these programs require inordinate amounts of CPU time to generate voltage and current waveforms over several hundred nanoseconds of simulated time. Simulation runs of several days on dedicated computers for critical circuit parts are not unheard of.

This situation spurred vigorous research aimed at reducing the cost of circuit simulation. This research attacked every aspect of the traditional circuit simulation problem, from the mathematical models used to represent integrated devices, to the underlying numerical integration and nonlinear equation solution algorithms, and the many program implementation details which had to be revisited in light of the advances in computer architecture, operating systems, and display technology.

The first major product of this research was the MOTIS program developed at Bell Labs [3]. Geared specifically towards digital MOS circuits, MOTIS was the first of the so-called *timing* simulators. It made possible for the first time the economical simulation of large circuits (up to 1000 logic gates) through simplifications in the traditional circuit simulation algorithm which took advantage of the unique characteristics of digital MOS circuits. MOTIS employs a simplified table-driven MOS device model, and calculates the output voltage for a typical logic gate using an approximate series/parallel current summation formula instead of the traditional circuit matrix solution method. Signals are propagated from gate to gate without iteration, as is done in logic simulators, using the Backward Euler integration formula with a small preset time step (typically 1 ns). The MOTIS algorithm produces voltage waveforms which are usually within 10 percent of detailed circuit simulation. Its accuracy, however, tends to deteriorate for circuits containing many bidirectional pass transistors and logical feedback loops. In fact, MOTIS may, without warning, produce erroneous results and even become numerically unstable [3].

Many of the techniques introduced in MOTIS were later used in the MOTIS-C program [5] developed at UC Berkeley. Unlike MOTIS, however, MOTIS-C uses Trapezoidal integration with a fixed time step computed automatically at the beginning of the analysis. In addition, MOTIS-C handles bidirectional pass transistors using a different decoupling process, and does not decouple the two simultaneous equations describing a floating capacitor. Overall, though, MOTIS-C has similar numerical properties and suffers from the same inaccuracies and instabilities as does MOTIS.

The next milestone came with the introduction of the concept of *mixed* circuit and logic simulation. This concept is based on the fact that a large portion of a typical digital circuit can be adequately modeled and simulated at the boolean logic gate level, while the rest of the circuit is modeled and simulated at the more detailed electrical (voltage and current) level. Mixed-level simulation can provide a speedup of up to two orders of magnitude over traditional circuit simulation with little loss in accuracy. DIANA [1], the first mixed-level simulator, achieves the goal of mixing circuit and logic models in the same simulation environment by dividing a network into a single "analog" block which interacts with boolean logic gate models through *threshold functions* and *boolean-controlled circuit elements*. SPLICE [14], another mixed-level simulator introduced about the same time, models a network as a collection of subnetworks each described either at the circuit or logic levels. The circuit level subnetworks are integrated with a common step size using the Backward Euler formula, and an algorithm similar to that in MOTIS and MOTIS-C is used to propagate signals among subnetworks. Interaction between adjacent subnetworks modeled at different levels is handled by explicitly inserting *thresholders* and *logic-to-voltage* and *logic-to-current converters*. An integer-time event scheduler is used to keep track of the *activity* of the various subnetworks as is

Manuscript received October 8, 1984; revised May 7, 1985.

K. A. Sakallah is with Digital Equipment Corporation, Hudson, MA 01749.

S. W. Director is with the Department of Electrical Engineering, Carnegie-Mellon University, Pittsburgh, PA 15213.

0278-0070/85/1000-0668$01.00

done in event-driven logic simulation. A later version of the SPLICE program, SPLICE1.7 [22], introduced the so called *iterated timing analysis* algorithm. This modification of the basic timing analysis algorithm, pioneered in the MOTIS program, ensures that the results of the analysis will be accurate by converging the subnetwork-to-subnetwork signal-propagation iteration. The modification also requires each node in the circuit to have a grounded capacitor. Performance results for the SPLICE1.7 program indicate speedups over SPICE2 which are about half those of the earlier noniterated implementation. In addition, the presence of floating capacitors slows the convergence of the method down considerably, causing it to be *slower* in some cases than standard circuit simulation.

The concept of latency at the circuit level was introduced in the MACRO program [16] developed at IBM. MACRO models a network as a collection of subnetworks which share a common integration time step. However, upon detecting that the time derivatives of the variables of a particular subnetwork are smaller than a given tolerance, and that the inputs to the subnetwork have not changed appreciably over the current time step, the subnetwork is considered latent and its equations are not solved.

Waveform relaxation was proposed in 1982 with the introduction of the RELAX program [11]. Waveform relaxation extends the well known Gauss-Seidel and Gauss-Jacobi iteration methods [4] from the linear and nonlinear algebraic equation domain to the nonlinear differential equation domain. Sufficient conditions for the convergence of waveform relaxation were given in [11] and shown to be satisifed by MOS circuits if each node in the circuit had a grounded capacitor. Initial performance results of the RELAX program, which was based on a node-by-node implementation of the waveform relaxation algorithm, showed speedups over SPICE2 ranging from 20 to 60 for several circuits. Further testing of the waveform relaxation method, however, indicated that for circuits containing logical feedback loops the convergence tends to be nonuniform and quite slow [28]. A modified version of the algorithm which partitions the simulation interval into "windows" which are small enough to allow faster convergence was subsequently implemented in the RELAX2 program [28]. RELAX2 also incorporates a simple circuit partitioning algorithm and relaxation is performed on a subnetwork-by-subnetwork basis.

This paper describes the *event driven circuit simulation* (EDCS) algorithm introduced in [19] and discusses its implementation in the SAMSON2 program. Section II reviews the traditional circuit simulation algorithm. Section III describes a scenario for EDCS and introduces the concepts which form the central theme of the paper. The mathematical details of the EDCS algorithm are elaborated in Sections IV (numerical integration) and V (decomposition). Section VI gives a brief description of the SAMSON2 program implementation. Several simulation examples are given in Section VII and conclusions in Section VIII.

II. Traditional Circuit Simulation

In order to delineate the main differences between the EDCS technique and traditional circuit simulation schemes, we begin with a quick overview of the major computational steps in traditional circuit simulation. Given the nonlinear system of algebraic-differential equations:

$$f(x, \dot{x}, t) = 0, \qquad x(0) = x_0 \tag{1}$$

which model the electrical circuit, the objective of traditional transient circuit simulation is to determine the approximate numerical solution at a sequence of time points. In this equation, x is an m-dimensional vector representing m circuit variables (e.g., voltages and currents), $\dot{x}$ is its time derivative, t represents time, and x_0 is a given initial condition. The desired numerical solution will be denoted by $\hat{x}(t_1), \hat{x}(t_2), \cdots, \hat{x}(t_n), \hat{x}(t_{n+1}), \cdots$. The caret '^' is used here to emphasize that these quantities are *numerically* computed. The *exact* solution of (1) will be denoted as $x(t_1), x(t_2), \cdots, x(t_n), x(t_{n+1}), \cdots$. When convenient, we will use the shorthand notation $\hat{x}_i$ and x_i to denote $\hat{x}(t_i)$ and $x(t_i)$, respectively.

The vector function f can be formulated in a variety of ways depending on what variables are included in the vector x. In Nodal Analysis, x is the vector of node voltages. In Modified Nodal Analysis [10], x is the vector of node voltages, as well as voltage-source and inductive-branch currents. A tableau formulation [9], is based on a vector x consisting of the node voltages, the branch voltages and currents, the capacitance charges, and the inductance fluxes. Despite this variety in equation formulation methods, the major steps in the traditional circuit simulation algorithm are fundamentally the same.

The solution at a specific time point, such as t_{n+1}, is determined in three stages. The first is a *discretization* stage which replaces the time derivatives $\dot{x}_{n+1}$ with divided difference approximations. This replacement transforms the circuit equations from an algebraic-differential into a purely algebraic system of equations. The second stage is a *nonlinear iteration* which solves the discretized circuit equations for the circuit variables $\hat{x}_{n+1}$. The third stage *estimates the error* in the just computed solution and decides whether to *accept* it or *reject* it and repeat the computation with a smaller step size. The following sections explain these three stages in some detail.

2.1. Discretization

In general circuit equations tend to be stiff[1], requiring a stiffly stable [6] implicit discretization function for stability reasons. Two of the most common implicit discretization functions used in circuit simulation are the second-order Trapezoidal Rule and the variable-order Backward Differentiation Formula (BDF) [12]. Several popular variants of the BDF formula exist including the Nordsieck vector form due to Gear [7], the backward difference form due to Brayton *et al.* [2], or the predic-

[1]By stiff we mean that the natural frequencies (time constants) of the circuit are widely separated.

tion-based form of Van Bokhoven [25]. Without loss of generality, and for reasons which will become obvious in Section IV, we restrict our attention in this paper to discretization using the BDF.

The kth-order BDF at t_{n+1} can be expressed most generally as

$$\hat{\dot{x}}_{n+1} = g_k(\hat{x}_{n+1}) \tag{2}$$

where k denotes the order of the formula. Note that the function g_k is implicit since it expresses the derivative of $\hat{x}$ at t_{n+1} in terms of $\hat{x}_{n+1}$. Note also that g_k depends on the k past values of $x - x_n, x_{n-1}, \cdots, x_{n-k+1}$—but this dependence is deliberately not shown because at t_{n+1} these k values are known quantities. Finally, note that the first-order formula, g_1, is the familiar Backward Euler method:

$$\hat{\dot{x}}_{n+1} = g_1(\hat{x}_{n+1}) = \frac{\hat{x}_{n+1} - x_n}{t_{n+1} - t_n}. \tag{3}$$

2.2. Nonlinear Iteration

At t_{n+1}, the circuit equations (1) become

$$f(x_{n+1}, \dot{x}_{n+1}, t_{n+1}) = 0. \tag{4}$$

Approximating $\dot{x}_{n+1}$ using (2), (4) reduces to a purely nonlinear algebraic equation:

$$f(\hat{x}_{n+1}, g_k(\hat{x}_{n+1}), t_{n+1}) \equiv F(\hat{x}_{n+1}) = 0 \tag{5}$$

which can be solved by iteration for $\hat{x}_{n+1}$.

The most common scheme for solving this nonlinear system of algebraic equations is the Newton–Raphson (NR) method which can be expressed as:

$$\nabla F(\hat{x}^i_{n+1})\, \hat{x}^{i+1}_{n+1} = -F(\hat{x}^i_{n+1}) + \nabla F(\hat{x}^i_{n+1})\, \hat{x}^i_{n+1}, \quad i = 0, 1, \cdots \tag{6}$$

where $\nabla F(\hat{x}^i_{n+1}) = \partial F/\partial \hat{x}_{n+1}]_{\hat{x}_{n+1} = \hat{x}^i_{n+1}}$ is the Jacobian matrix of F, and i is an iteration counter. Starting from an initial guess $\hat{x}^0_{n+1}$, (6) will, under certain conditions, generate a sequence $\hat{x}^1_{n+1}, \hat{x}^2_{n+1}, \cdots$ which converges to the solution $\hat{x}_{n+1}$. Thus the NR method transforms the problem of solving the nonlinear system of equations (5) into that of solving a sequence of linear systems. In general the Jacobian matrix ΔF is sparse and the linear system of equations represented by (6) is solved using sparse Gaussian Elimination [17].

2.3. Truncation Errors

Assuming that the numerical error introduced by the nonlinear iteration (6) is negligible, the main source of error in the solution $\hat{x}_{n+1}$ is due to the discretization function g_k. The function g_k is essentially a truncated series approximation to the derivative. The error in the solution, therefore, is a truncation error. The estimation and control of *local* truncation error (LTE), i.e., truncation error introduced in a single discretization step, is the key to the practical implementation of circuit simulation programs.

Two types of LTE estimates are usually used in circuit simulation. The first type, called *a posteriori* LTE estimates, are expressed as a function of the just computed numerical solution. The second type, called *a priori* LTE estimates, are expressed as a function of a step size $h > 0$ and are calculated before the numerical solution is found. *A posteriori* estimates are more accurate, and are used as the basis for accepting or rejecting the numerical solution. *A priori* estimates, on the other hand, are used to automatically select the size of the next discretization step which keeps the LTE below a user-specified error tolerance.

2.4. Mathematical Definitions

To facilitate the description of the circuit simulation algorithms in this paper we need to define several vector operations and then introduce three vector error operators. In these definitions, x and y are considered to be the following m-dimensional vectors:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_m \end{pmatrix} \qquad y = \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{pmatrix}.$$

Definition 2.1: The *absolute value* of the m-dimensional vector x is an m-dimensional vector denoted by $|x|$ and defined as

$$|x| = \begin{pmatrix} |x_1| \\ |x_2| \\ \cdots \\ |x_m| \end{pmatrix}$$

Definition 2.2: The *maximum value* of the two m-dimensional vectors x and y is an m-dimensional vector denoted by $\max(x, y)$ and defined as

$$\max(x, y) = \begin{pmatrix} \max(x_1, y_1) \\ \max(x_2, y_2) \\ \cdots \\ \max(x_m, y_m) \end{pmatrix}.$$

Definition 2.3: The m-dimensional vector x is said to be *less than or equal to* (*greater than*) the m-dimensional vector y, expressed as $x \leq y$ ($x > y$), if and only if $x_i \leq y_i$ ($x_i > y_i$) for $i = 1, 2, \cdots, m$.

In the following definition we assume that the vector x is a function of the positive scalar variable $\lambda \geq 0$, i.e.,

$$x(\lambda) = \begin{pmatrix} x_1(\lambda) \\ x_2(\lambda) \\ \cdots \\ x_m(\lambda) \end{pmatrix}.$$

Definition 2.4: The solution λ^* of the vector equation:

$$x(\lambda) = y$$

Comments: t_n denotes the last successful simulation time point
t_{n+1} denotes the current simulation time point
$h_{\min}$ denotes the smallest allowable integration step size (given)
x_0, the initial condition (given

Step	Action
0 (Initialization)	$t_n = 0,\ t_{n+1} = h_{\min}$
1 (Time Loop)	**repeat**
2 (Discretization)	$\hat{x}_{n+1} = g_k(\hat{x}_{n+1})$
3 (Nonlinear Iteration)	Solve $f(\hat{x}_{n+1}, g_k(\hat{x}_{n+1}), t_{n+1}) = 0$ for $\hat{x}_{n+1}$
4 (Step Acceptance)	**if** $\lvert E^x_{n+1} \rvert \leq T^x_{n+1}$
4a (Update Time Grid)	$t_n = t_{n+1}$
4b (Next Step Forward)	Solve $\lvert \mathcal{E}^x_n(h) \rvert = T^x_n$ for $h = h_{\text{next}}$
5 (Step Rejection)	**else**
5a (Size of Repeat Step)	Solve $\lvert \hat{\mathcal{E}}^x_n(h) \rvert = T^x_{n+1}$ for $h = h_{\text{next}}$
	endif
6 (Update Time)	$t_{n+1} = t_n + h_{\text{next}}$
7 (End Time Loop)	**until** $t_{n+1} > t_f$

Algorithm 2.1: Traditional Circuit Simulation Algorithm for solving (1) from $t = 0$ to $t = t_f$.

is

$$\lambda^* = \min(\lambda_1^*, \lambda_2^*, \cdots, \lambda_m^*)$$

where λ_i^* for $i = 1, 2, \cdots, m$ is the solution of the scalar equation:

$$x_i(\lambda) = y_i.$$

Definition 2.5: Vector Error Operators

A posteriori Vector LTE Operator:

$$E^x_n \equiv x(t_n) - \hat{x}(t_n)$$

A priori Vector LTE Operator:

$$\mathcal{E}^x_n(h) \equiv x(t_n + h) - \hat{x}(t_n + h), \qquad h > 0$$

Vector Error Tolerance Operator:

$$T^x_n \equiv \epsilon_a + \epsilon_r \max(\lvert \hat{x}(t_n) \rvert, \lvert \hat{x}(t_{n-1}) \rvert)$$

where ϵ_r is a relative error parameter and ϵ_a is a vector of absolute error parameters.

2.5. Circuit Simulation Algorithm

Algorithm 2.1 outlines the major steps of traditional transient circuit simulation. These steps are further explained by the following remarks:

1) The initial step size is set to a user-provided minimum step size $h_{\min}$ (Step 0) because on startup there is no other information available to determine a suitable step size. In addition, $h_{\min}$ serves to prevent the time step selection mechanism from getting into a non-ending loop of time step reductions (Step 5) if the user-specified error parameters are too tight.
2) The LTE test in Step 4 is a vector comparison (see Definition 2.3) which will fail if one or more of the components in x has an LTE which exceeds the specified tolerance. In addition to this test, the robustness of the algorithm is usually enhanced by including a mechanism which rejects the solution if convergence of the nonlinear iteration in Step 3 is not attained within a specified number of iterations, such as 10.
3) The step size calculation formulas in Steps 4b and 5a are vector formulas (see Definition 2.4). The solution h_{next} must, therefore, be the *minimum* of the m positive solutions of the m component formulas. In addition, it has been found that the number of step rejections can be minimized if the step size determined from these truncation error formulas is limited to no more than twice the size of the previous step.
4) A time step is rejected if the *a priori* LTE estimate used to calculate it turns out to be an underestimate of the *a posteriori* LTE. The size of the repeat step in this case is calculated using a *revised* estimate of the *a priori* error, denoted as $\hat{\mathcal{E}}^x_n(h)$ in Step 5a, which reflects the latest available *a posteriori* error information. Most existing circuit simulators use simpler schemes to select the size of a repeat step. A common scheme is to choose a repeat step size which is one-fourth or one-eighth the size of the current step. As will be shown in Section 4.2.3, the method indicated in the above algorithm is more rigorous. In addition, experimental evidence indicates that it is more efficient.
5) To avoid complicating the discussion unnecessarily, the step size selection scheme shown in the above algorithm does not address changing the order k of the discretization function. In general, the step size

calculation formulas are solved for the current order k and for orders $k-1$ and $k+1$ yielding three different step sizes. Discretization over the next interval is then done using the order which yielded the largest step size.

6) Note that Step 6 will advance simulation time if the current time step was accepted since t_n is updated in Step 4a. On the other hand, simulation time will be retracted in Step 6 if the current time step was rejected.

III. A Scenario for Event Driven Circuit Simulation

The motivation for EDCS can be found by examining the step size selection scheme in Algorithm 2.1 (Steps 4b and 5a). This scheme selects a single step size to discretize all the components of the unknown vector x. In general, the size of the integration step for a single component of x reflects the rate of change of that component [19]. Specifically, a fast-changing component requires a smaller step size than a slower changing component. Thus the use of a single step size for all components of x exacts an unnecessarily high computational cost in large "*temporally sparse*" networks by needlessly solving for variables which are changing at a much slower rate than the fastest variables. A network is considered temporally sparse if, at any instant of time, there is a wide spread in the rates of change of its variables.

3.1. Network Partitioning

The EDCS algorithm attempts to take advantage of temporal sparseness by appropriately bypassing the evaluation and solution of certain subsets of the network equations (1) without impacting the overall accuracy of the analysis. We propose to do this by *partitioning* the network into σ loosely interconnected multi-terminal subnetworks [19] and choosing a separate step size for each subnetwork. Specifically, we consider that the network equations (1) can be written as σ sets of *subnet* equations and σ sets of *connection* equations:

$$\textit{Subnet Eqns:} \qquad f_i(w_i, \dot{w}_i, y_i, u_i) = 0, \quad i = 1, \cdots, \sigma \tag{7}$$

$$\textit{Connection Eqns:} \qquad \Theta_i y + u_i = \eta_i, \quad i = 1, \cdots, \sigma \tag{8}$$

where the variables of the ith subnetwork are divided into three sets: inputs u_i, outputs y_i, and internal variables w_i. The subnet equations f_i are nonlinear algebraic-differential equations which can be formulated using any of the methods mentioned in Section II. The particular formulation method chosen will determine which of the internal subnet variables are included in the vector w_i. The input and output variables, on the other hand, are basically the terminal voltages and currents of the subnetwork such that, at any given terminal, if the voltage is designated as the input variable then the current is the corresponding output variable and vice versa. It is important to note that this classification of terminal variables does not imply any directionality assumptions on the subnetwork but is merely a notational convenience.

The connection equations are the Kirchhoff voltage and current laws at the nodes where terminals of adjacent subnetworks are connected. They express the constraints of interconnection among the subnetworks in terms of their input and output variables u_i and y_i. In these equations, Θ_i are topological $(0, \pm 1)$ fan-in matrices, η_i are vectors of externally specified inputs, and $y = \text{col}(y_1, y_2, \cdots, y_\sigma)$ is the vector of all output variables.

Notice how these partitioned network equations permit the subnet equations of the ith subnetwork to be solved for the internal and output variables w_i and y_i independently of the subnet equations of other subnetworks once the inputs u_i are specified. Thus all interaction among the subnetworks is through their terminal variables via the connection equations.

3.2. Alert and Dormant Models

The numerical solution of the ith subnetwork is generated at a sequence of time points $t_1^i, t_2^i, \cdots, t_n^i, t_{n+1}^i, \cdots$ which may be distinct from the time points of other subnetworks. This sequence can be viewed as a sequence of *events* for the ith subnetwork. At each of these events, the subnetwork is considered to be *alert*; between these events it is considered to be *dormant*.

When a subnetwork is alert, it is modeled by its nonlinear algebraic-differential system of subnet equations f_i, which will be called the "*alert model.*" When dormant, the subnetwork is modeled by an approximate set of output *extrapolation* equations called the "*dormant model.*" The dormant model is closely related to the discretization process used to solve the alert model. In particular, the dormant model associated with a discretization function of order k_i is a polynomial of order $(k_i + 1)$ which can be expressed as follows:

$$y_i^D(t_n^i + h) \equiv \sum_{j=1}^{k_i+1} a_j h^j \tag{9}$$

where we use the superscript D to indicate a dormant model.

This dormant model essentially *decouples* the subnetwork from the rest of the network. In addition, use of a dormant model avoids the discretization and linearization steps required to solve an alert model. Hence, the substitution of a dormant model for an alert model generally leads to a reduction in computation time. However, since the dormant model is only an approximation to the alert model, its applicability must be checked by monitoring the inputs to the subnetwork to ensure that they do not change excessively. Detection of appreciable deviation of the inputs from their expected values is an indication that the dormant model, which is based on the past behavior of the subnetwork, is no longer valid. To avoid significant *decoupling* errors when this situation is detected, the sub-

network is made alert, i.e., we reject the current (yet uncompleted) time step and replace the dormant model with the alert model using a reduced step size.

3.3. Event Scheduling

The use of separate step sizes for different subnetworks makes it necessary to employ a mechanism for scheduling events and updating simulation time. A detailed discussion of event scheduling techniques can be found in [19]. For our purposes, it will be sufficient to describe the main scheduling functions needed in the proposed EDCS algorithm.

We begin with a formal definition of events.

Definition 3.1: An event e is a 2-tuple (t, i) where t is the *event time* and i is the index of a subnetwork which becomes alert at that time.

The three scheduling functions required in the algorithm are:

schedule (e): Places the event e in a chronologically-ordered list of events, commonly called the event queue. This function must be able to handle future as well as past events. A future event is one whose event time is greater than the current simulation time. A past event, on the other hand, is defined to be an event whose event time is smaller than the current simulation time.[2]

cancel (e): Removes the event e from the event queue.

next events (A, t): Searches the event queue for the event or events with the earliest event time, removes them from the queue, and returns that time in the variable t. This function also returns, in the set A, the indices of the subnetworks corresponding to the removed events.

3.4. Event Driven Circuit Simulation Algorithm

Algorithm 3.1 outlines the major steps of the proposed event driven circuit simulation scheme. It assumes the following extension of the earlier definition of error operators (Definition 2.5):

Definition 3.2: Vector Error Operators

A posteriori Vector LTE Operator:

$$E_n^{x_i} \equiv x_i(t_n^i) - \hat{x}_i(t_n^i).$$

A priori Vector LTE Operator:

$$\mathcal{E}_n^{x_i}(h) \equiv x_i(t_n^i + h) - \hat{x}_i(t_n^i + h), \qquad h > 0$$

Vector Error Tolerance Operator:

$$T_n^{x_i} \equiv \epsilon_a^i + \epsilon_r \max(|\hat{x}_i(t_n^i)|, |\hat{x}_i(t_{n-1}^i)|).$$

Remarks:

1) The structure of this algorithm is essentially the same as that of Algorithm 2.1. In fact, Algorithm 3.1 reduces to Algorithm 2.1 if the network consists of a single subnetwork ($\sigma = 1$).

2) At any given time, the σ subnetworks are classified into an alert set A and a dormant set D. Initially, A will be the set of all σ subnetworks and D will be the empty set (Step 0). The simulation is started with all subnetworks taking the minimum step $h_{\min}$.

3) The computational core of the algorithm is the nonlinear iteration in Step 5. This iteration solves a system of equations consisting of the discretized equations of alert subnetworks (Step 2), the output extrapolation equations of dormant subnetworks (Step 3), and the connection equations of all subnetworks. Note that this iteration calculates all the network variables except for the internal and output variables of dormant subnetworks.

4) Upon convergence of the nonlinear iteration, the truncation errors of the input variables of dormant subnetworks are computed and compared with specified tolerances (Step 6). If this *dormancy check* fails, i.e., if the errors in the inputs exceed the tolerances, the subnetwork is alerted by canceling its future event, removing it from the dormant set, and adding it to the alert set (Step 6a). The subnet equations of the alerted subnetwork are then discretized at the current simulation time (Step 6b). A new set of equations containing more alert and less dormant models is now set up and solved (Step 5). This process is repeated until all remaining dormant subnetworks satisfy the dormancy test (Step 7).

5) Step size control is essentially the same as in the traditional circuit simulation algorithm. The truncation errors of the internal variables for each alert subnetwork are calculated and compared to specified tolerances. If these errors are less than the corresponding tolerances (Step 8), the time grid of that subnetwork is updated (Step 8a) and the size of the next step, h_{next}^i, is computed based on an estimate of the *a priori* LTE of the internal variables (Step 8b). If, on the other hand, the errors exceed the tolerances (Step 9), the size of a repeat step h_{next}^i is calculated using revised estimates of these a priori errors (Step 9a). In either case, h_{next}^i is used to calculate the time t_{n+1}^i at which the subnetwork should become alert again, and an event is scheduled at that time (Step 10). Simulation time t is updated by calling *nextevents*, new sets of alert and dormant subnetworks are established (Step 11), and the whole process is repeated (from Step 1) until the end of the simulation intervals t_f is reached or there are no more events in the event queue (Step 12).

6) Careful examination of Algorithm 3.1 reveals that when the LTE test fails for a specific subnetwork, only that subnetwork's time step is rejected and repeated; the "good" time steps of adjacent subnetworks are not discarded. Underlying this behavior, which is a consequence

[2]By generating past events when a time step is rejected and repeated because of excessive truncation error, we essentially have a mechanism for retracting simulation time. In effect, the concept of past events generalizes the traditional event scheduling methods.

Comments: A is the set of *alert* subnetworks
D is the set of *dormant* subnetworks
t denotes current simulation time
$h_{\min}$ denotes the smallest allowable integration step size (given)

Step	Action
0 (Initialization)	$A = \{1, 2, \cdots, \sigma\}$, $D = \varnothing$, $t = h_{\min}$, $t^i_{n+1} = h_{\min}$ for $i \in A$
1 (Time Loop)	**repeat**
2 (Discretization)	$\hat{w}_i = g_{k_i}(\hat{w}_i)$, for $i \in A$
3 (Extrapolation)	$\hat{y}_i = y^D_i$, for $i \in D$
4 (Dormancy Loop)	**repeat**
5 (Nonlinear Iteration)	Solve $f_i(\hat{w}_i, g_{k_i}(\hat{w}_i), \hat{y}_i, \hat{u}_i) = 0$, for $i \in A$; $\hat{y}_i = y^D_i$, for $i \in D$; $\Theta_i \hat{y} + \hat{u}_i = \eta_i$, $\forall i$
6 (Dormancy Check)	**if** $\lvert E^{u_i}_{n+1} \rvert > T^{u_i}_{n+1}$, for $i \in D$
6a (Activation)	*cancel* $((i, t^i_{n+1}))$, $t^i_{n+1} = t$, $D = D - \{i\}$, $A = A + \{i\}$
6b (Discretization)	$\hat{w}_i = g_{k_i}(\hat{w}_i)$
	endif
7 (End Dormancy Loop)	**until** $\lvert E^{u_i}_{n+1} \rvert \le T^{u_i}_{n+1}$, for $i \in D$
8 (Step Acceptance)	**if** $\lvert E^{w_i}_{n+1} \rvert \le T^{w_i}_{n+1}$, for $i \in A$
8a (Update Time Grid)	$t^i_n = t^i_{n+1}$
8a (Next Step Forward)	Solve $\lvert \mathcal{E}^{w_i}_n(h) \rvert = T^{w_i}_n$, for $h = h^i_{\text{next}}$
9 (Step Rejection)	**else**
9a (Size of Repeat Step)	Solve $\lvert \hat{\mathcal{E}}^{w_i}_n(h) \rvert = T^{w_i}_{n+1}$, for $h = h^i_{\text{next}}$
	endif
10 (Scheduling)	$t^i_{n+1} = t^i_n + h^i_{\text{next}}$, *schedule* $((i, t^i_{n+1}))$, for $i \in A$
11 (Time Control)	*nextevents* (A, t), $D = \{1, 2, \cdots, \sigma\} - A$
12 (End Time Loop)	**until** $t > t_f$ or event queue is empty

Algorithm 3.1: Event Driven Circuit Simulation Algorithm for solving (7) and (8) from $t = 0$ to $t = t_f$.

of the event driven nature of the algorithm, are the following two assumptions:

(a) The growth in LTE of the internal variables w_i of a subnetwork is due mainly to the internal dynamics of that subnetwork. Interaction with other subnetworks has only a second order effect on such LTE growth.

(b) The dormant extrapolation model remains accurate over the calculated time step and need not be discarded even if the time step had to be repeated due to excessive LTE growth of the internal variables.

In practice these two assumptions are not unreasonable. Specifically, the dormancy check (Step 6) ensures that a dormant subnetwork will be activated before the expiration of its dormant period, thereby avoiding a possible step rejection at the end of that period. Thus, this check effectively minimizes the incidence of time step rejections. In addition, the fact that the chosen step size is never allowed to be more than twice the size of the previous step ensures the accuracy of the extrapolation model in case that particular time step does get rejected. Experience with the SAMSON2 program indicates that this step rejection mechanism works quite well in practice.

7) Whereas Step 5 (nonlinear iteration) implies that the connection equations of all subnetworks (both alert and dormant) are evaluated and solved at every time point, in practice only a subset of these equations needs to be solved. The details of such a *bypass* scheme are explained in Section 5.2.

8) Experimentally it was observed that dormant subnetworks with event times which are only slightly larger than the current simulation time tend to almost always be activated. To minimize the activation overhead, a *gravity* mechanism which attracts *imminent* events is incorporated in the event scheduler. In this context, an event is considered imminent if its event time is within a specified mul-

tiple of $h_{\min}$ from current simulation time. While experimental evidence is still inconclusive, a "gravity field" equal to a single minimum step size $h_{\min}$ seems to work well in practice.

9) Historically, the motivation for EDCS was inspired by event driven logic simulation techniques [19]. Bearing in mind the slight extensions to the traditional scheduling functions used in logic simulators, the structure of Algorithm 3.1 is in fact identical to that of event driven logic simulators. This fact is used in [19] to devise a mixed level circuit and logic simulation algorithm.

10) An interesting way to view the alternating use of alert and dormant models for a given subnetwork is to consider which variables are used to control the growth of truncation error during each phase. When alert, the truncation errors of the internal variables $E^{w_i}_{n+1}$ are used; when dormant, the truncation errors of the input variables $E^{u_i}_{n+1}$ are used.

In order to implement the above algorithm we must specify the various formulas used for discretization, *a priori*, *a posteriori*, and decoupling errors, and the decomposition scheme used to solve the alert subset of the network equations. Discussion of these topics follows.

IV. Numerical Integration

As can be seen from Algorithm 3.1, in order for a numerical integration scheme to be useable for event driven circuit simulation it must satisfy the following criteria.

1) The discretization formula must be stiffly stable.
2) The error estimates must be easy to calculate.
3) The scheme must allow for efficient extrapolation of variable values to construct the dormant model.

All three formulations of the BDF mentioned in Section II as well as the Trapezoidal Rule satisfy the first two criteria, but only the prediction-based differentiation (PBD) formulation of the BDF satisfies the last criterion.

In this section we state the PBD formulas without derivation (interested readers may refer to [25], [19]) and derive estimates for the local truncation and decoupling errors as well as the dormant model equations. We conclude the section by illustrating with an example how these formulas are used in the EDCS algorithm.

4.1. PBD Formulas

The PBD formulas of order k are defined over a nonuniform time grid $\{t_{n-k}, t_{n-k+1}, \cdots, t_n, t_{n+1}\}$ where t_{n+1} represents current time and t_n is the most recent time point at which an acceptable solution was found. To simplify subsequent manipulations, we introduce the following time grid functions:

$$h_i = t_{n+1} - t_{n+1-i} \qquad h'_i = t_n - t_{n-i}$$

$$\alpha_i = \prod_{j=1}^{i} h_j \qquad \alpha'_i = \prod_{j=1}^{i} h'_j$$

$$\beta_i = \sum_{j=1}^{i} \frac{1}{h_j} \qquad \beta'_i = \sum_{j=1}^{i} \frac{1}{h'_j}. \tag{10}$$

The PBD formulas can now be succinctly stated as follows:

$$x_{n+1} = \bar{x}^l_{n+1} + R_l, \qquad l = 1, 2, \cdots, k+1 \tag{11}$$

$$\dot{x}_{n+1} = \beta_k x_{n+1} - \bar{x}^k_{n+1} + R'_{k+1} \tag{12}$$

where

$$R_l = \alpha_l \frac{x^{(l)}(\zeta)}{l!}, \qquad t_{n+1-l} \le \zeta \le t_{n+1} \tag{13}$$

$$R'_{k+1} = \alpha_k \frac{x^{(k+1)}(\xi)}{(k+1)!}, \qquad t_{n+1-k} \le \xi \le t_{n+1} \tag{14}$$

$$\bar{x}^k_{n+1} \equiv \sum_{i=1}^{k} \frac{\bar{x}^i_{n+1}}{h_i} \tag{15}$$

and $\bar{x}^l_{n+1}$ is defined recursively from

$$\bar{x}^1_{n+1} = x_n$$

$$\bar{x}^{i+1}_{n+1} = \bar{x}^i_{n+1} + \frac{\alpha_i}{\alpha'_i}(x_n - \bar{x}^i_n), \qquad i = 1, \cdots, k. \tag{16}$$

In these formulas, R_l and R'_{k+1} represent high-order remainder terms, and $\bar{x}^l_{n+1}$ is the value at t_{n+1} of the $(l-1)$th order polynomial passing through the past l points $x_n, x_{n-1}, \cdots, x_{n+1-l}$. Assuming that $x^{(k+1)}(t)$ is constant for $t_{n-k} \le t \le t_{n+1}$, the remainders can be shown to be related as follows:

$$R'_{k+1} \approx \frac{\alpha_k}{\alpha_{k+1}} R_{k+1} = \frac{x_{n+1} - \bar{x}^{k+1}_{n+1}}{h_{k+1}}. \tag{17}$$

Note that for $k = 1$, (12) reduces to

$$\dot{x}_{n+1} = \frac{x_{n+1} - x_n}{h_1} + h_1 \frac{\ddot{x}(\xi)}{2!}, \qquad t_n \le \xi \le t_{n+1} \tag{18}$$

which is the familiar Backward Euler integration formula with a remainder.

4.2. Step-Size Control

We now consider the *a posteriori* and *a priori* local truncation error estimates associated with the PBD formulas. Since these estimates do not depend on how the network equations are partitioned, we will develop them assuming that the network can be modeled by (1), repeated here for convenience:

$$f(x, \dot{x}, t) = 0 \tag{19}$$

where we assume x to be an m-vector.

4.2.1. A Posteriori Local Truncation Error

At time t_{n+1}, the *exact* solution x_{n+1} satisfies

$$f(x_{n+1}, \beta_k x_{n+1} - \bar{x}^k_{n+1} + R'_{k+1}, t_{n+1}) = 0. \tag{20}$$

On the other hand, the *numerical* solution $\hat{x}_{n+1}$ satisfies

$$f(\hat{x}_{n+1}, \beta_k \hat{x}_{n+1} - \bar{x}^k_{n+1}, t_{n+1}) = 0 \tag{21}$$

Expanding (20) in a Taylor series about $\hat{x}_{n+1}$ we obtain:

$$\left[\left(\frac{\partial f}{\partial \dot{x}}\right)^{-1}\left(\frac{\partial f}{\partial x}\right) + \beta_k I\right](x_{n+1} - \hat{x}_{n+1}) = -R'_{k+1} \tag{22}$$

where the partial derivatives are evaluated at a point between the exact and numerical solutions.

If we now assume that

$$\left\| \left(\frac{\partial f}{\partial \dot{x}}\right)^{-1} \left(\frac{\partial f}{\partial x}\right) \right\|_{\infty} \ll \beta_k$$

equation (22) reduces to

$$\beta_k(x_{n+1} - \hat{x}_{n+1}) = -R'_{k+1}. \tag{23}$$

Finally, substituting for R'_{k+1} from (17), and invoking the definition of the a posteriori error operator, we obtain

$$E^x_{n+1} = x_{n+1} - \hat{x}_{n+1} \approx \frac{\bar{x}^{k+1}_{n+1} - \hat{x}_{n+1}}{h_{k+1}\beta_{k+1}}. \tag{24}$$

The LTE at t_{n+1} is thus seen to be proportional to the difference between the kth order extrapolation of x_{n+1} (a predictor) and the computed value of x_{n+1} (a corrector.)

4.2.2. A Priori Local Truncation Error

To establish the *a priori* error estimate $\mathcal{E}^x_n(h)$ we must determine the functional dependence of local truncation error on the step size. We start by noting that (23) leads to the following expression for E^x_{n+1}:

$$E^x_{n+1} = -\frac{R'_{k+1}}{\beta_k} = -\frac{\prod_{i=1}^{k} h_i}{\sum_{i=1}^{k} \frac{1}{h_i}} \frac{x^{(k+1)}(\xi)}{(k+1)!}, \qquad t_{n+1-k} \le \xi \le t_{n+1} \tag{25}$$

which shows the explicit dependence of the error at t_{n+1} on the step sizes h_i. The projected error at $(t_n + h)$ can be obtained from (25) by recognizing that $h_i = h_1 + h'_{i-1}$, and by substituting h for h_1:

$$x(t_n + h) - \hat{x}(t_n + h) = -\frac{\prod_{i=1}^{k} (h + h'_{i-1})}{\sum_{i=1}^{k} \frac{1}{(h + h'_{i-1})}} \frac{x^{(k+1)}(\hat{\xi})}{(k+1)!} \tag{26}$$

where $t_{n+1-k} \le \hat{\xi} \le t_n + h$.

To arrive at the desired expression for *a priori* LTE we must consider two cases. In the first case, the just computed solution $\hat{x}_{n+1}$ has been accepted and the time grid has been updated so that $t_n = t_{n+1}$. The *a posteriori* LTE at the last good time point, t_n, can, therefore, be expressed as (see (25)):

$$E^x_n = -\frac{\prod_{i=1}^{k} h'_i}{\sum_{i=1}^{k} \frac{1}{h'_i}} \frac{x^{(k+1)}(\xi)}{(k+1)!}, \qquad t_{n-k} \le \xi \le t_n. \tag{27}$$

The *a priori* estimate in this case can now be obtained by substituting $x^{(k+1)}(\xi)$ from (27) for $x^{(k+1)}(\hat{\xi})$ in (26):

$$\mathcal{E}^x_n(h) = x(t_n + h) - \hat{x}(t_n + h) \approx \frac{\prod_{i=1}^{k} (h + h'_{i-1})}{\sum_{i=1}^{k} \frac{1}{(h + h'_{i-1})}} \frac{\beta'_k}{\alpha'_k} E^x_n. \tag{28}$$

In the second case, the just computed solution $\hat{x}_{n+1}$ has been rejected and the time grid was not modified (i.e., t_n was not advanced). A *revised a priori* LTE estimate, denoted by $\hat{\mathcal{E}}^x_n(h)$ to distinguish it from the estimate $\mathcal{E}^x_n(h)$ in (27), can now be obtained by substituting $x^{(k+1)}(\xi)$ from (25) for $x^{(k+1)}(\hat{\xi})$ in (26):

$$\hat{\mathcal{E}}^x_n(h) = x(t_n + h) - \hat{x}(t_n + h) \approx \frac{\prod_{i=1}^{k} (h + h'_{i-1})}{\sum_{i=1}^{k} \frac{1}{(h + h'_{i-1})}} \frac{\beta_k}{\alpha_k} E^x_{n+1}. \tag{29}$$

Note that these two *a priori* estimates have the same functional dependence on the step size h and differ only by scale factors which reflect the fact the time grid was updated in the case of $\mathcal{E}^x_n(h)$ and was not updated in the case of $\hat{\mathcal{E}}^x_n(h)$.

4.2.3 Step-Size Selection

The acceptability of the numerical solution $\hat{x}_{n+1}$ is determined from the following inequality:

$$|E^x_{n+1}| \le T^x_{n+1} \tag{30}$$

where E^x_{n+1} is the *a posteriori* LTE estimate given by (24).

If the inequality is satisfied, the computed solution $\hat{x}_{n+1}$ is accepted and the time grid is updated so that $t_n = t_{n+1}$. The next time point, $t_{\text{next}} \equiv t_n + h_{\text{next}}$, is now determined by solving for the step size h_{next} from:

$$|\mathcal{E}^x_n(h)| = T^x_n, \qquad h > 0 \tag{31}$$

where $\mathcal{E}^x_n(h)$ is the *a priori* estimate given by (28).

If, on the other hand, the inequality is not satisfied, the computed solution $\hat{x}_{n+1}$ is rejected and t_{next} is found by solving for h_{next} from

$$|\hat{\mathcal{E}}^x_n(h)| = T^x_{n+1}, \qquad h > 0$$

where $\hat{\mathcal{E}}^x_n(h)$ is a revised estimate of the *a priori* LTE as given by (29) which reflects a faster error growth rate than originally anticipated.

This step selection scheme is illustrated graphically in Fig. 1 for an integration order $k = 1$.

4.3. The Dormant Model

The dormant model describes the expected behavior of the vector x in the time interval $t_n \le t < t_n + h_{\text{next}}$. Its derivation is based on the same assumptions which led to the a priori truncation error estimates in (28) and (29).

We start from (11) with t_{n+1} replaced by $(t_n + h)$ and with $l = k + 1$:

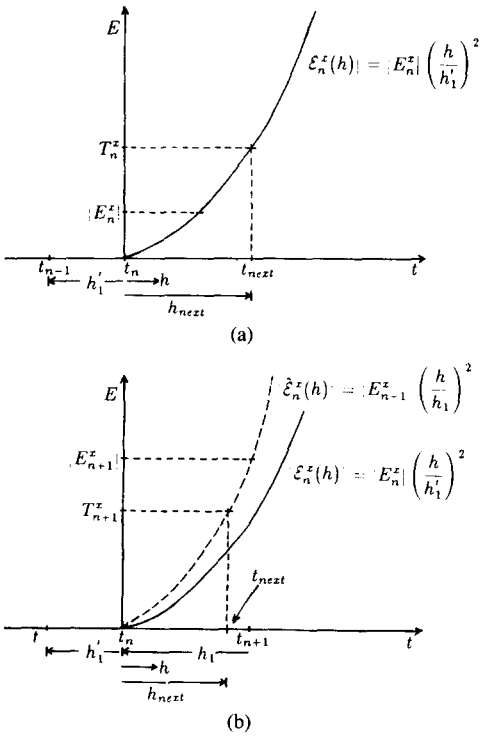


Fig. 1. Step size selection process. (a) Just after numerical solution was *accepted*. (b) Just after numerical solution was *rejected*.

$$x(t_n + h) = \bar{x}^{k+1}(t_n + h) + \prod_{i=1}^{k+1} (h + h'_{i-1}) \frac{x^{(k+1)}(\hat{\xi})}{(k+1)!} \tag{33}$$

where $t_{n-k} \leq \hat{\xi} \leq t_n + h$ and the definition of the remainder is invoked from (13). The kth-order predictors $\bar{x}^{k+1}(t_n + h)$ above are given by (see (16)):

$$\bar{x}^{k+1}(t_n + h) = x_n + \sum_{i=1}^{k} \left[\prod_{j=1}^{i} \left(\frac{h + h'_{j-1}}{h'_j} \right) \right] (x_n - \bar{x}_n^i). \tag{34}$$

The desired dormant model, which we denote by $x^D(t_n + h)$, can be obtained from (33) by substituting $x^{(k+1)}(\xi)$ from (27) for $x^{(k+1)}(\hat{\xi})$:

$$x^D(t_n + h) \approx \bar{x}^{k+1}(t_n + h) - \frac{\beta'_k}{\alpha'_k} E_n^x \prod_{i=1}^{k+1} (h + h'_{i-1}) \tag{35}$$

where $0 \leq h \leq h_{\text{next}}$. Note that the dormant model is essentially a kth-order predictor of x with a $(k+1)$th-order correction term which involves the *a posteriori* error E_n^x at the last good time point.

4.4. An Example

The following example illustrates the use of the formulas derived above in the EDCS algorithm. For simplicity we assume the order k to be 1, and consider a single subnetwork which has exactly one input u, one output y, and one internal (differential) variable w. The solution $(\hat{w}_{n+1}, \hat{y}_{n+1}, \hat{u}_{n+1})$ is found by substituting for $\dot{\hat{w}}_{n+1}$ in the subnetwork equations from (18):

$$\dot{\hat{w}}_{n+1} = \frac{\hat{w}_{n+1} - w_n}{h_1}. \tag{36}$$

The local truncation error E_{n+1}^w of the variable w is now calculated from (24):

$$E_{n+1}^w = \frac{w_n + (h_1/h_1')(w_n - w_{n-1}) - \hat{w}_{n+1}}{1 + h_2/h_1}. \tag{37}$$

If this error exceeds the tolerance T_{n+1}^w, the solution $\hat{x}_{n+1}$ is rejected and a repeat step is calculated by substituting the revised *a priori* error estimate:

$$\hat{\mathcal{E}}_n^w(h) \approx E_{n+1}^w (h/h_1)^2 \tag{38}$$

in (32) yielding

$$h_{\text{next}} = h_1 \sqrt{\frac{T_{n+1}^w}{|E_{n+1}^w|}}. \tag{39}$$

If, however, the error E_{n+1}^w is less than or equal to the tolerance T_{n+1}^w, the computed solution $\hat{x}_{n+1}$ is accepted, $t_n = t_{n+1}$, and the size of the next step is now determined by substituting the *a priori* error estimate:

$$\mathcal{E}_n^w(h) \approx E_n^w (h/h_1')^2 \tag{40}$$

in (31) yielding

$$h_{\text{next}} = h_1' \sqrt{\frac{T_n^w}{|E_n^w|}}. \tag{41}$$

While the subnetwork is dormant (during $t_n \leq t < t_n + h_{\text{next}}$) it interacts *passively* with its environment. Its output y is approximated by

$$y(t_n + h) \approx y^D(t_n + h) = y_n + (y_n - y_{n-1}) \frac{h}{h_1'} - E_n^y \frac{h}{h_1'} \left(1 + \frac{h}{h_1'}\right) \tag{42}$$

where E_n^y is given by an equation similar to (36). At the same time, the input u, which is computed whenever a neighboring subnetwork becomes alert, is monitored. The computed value of this input, $\hat{u}(t_n + h)$, is used to calculate the LTE estimate:

$$E^u(t_n + h) = \frac{u_n + (u_n - u_{n-1})(h/h_1') - \hat{u}(t_n + h)}{(2 + h_1'/h)} \tag{43}$$

which is viewed as a decoupling error. The dormancy check compares this error to the tolerance T_{n+1}^u and causes the subnetwork to be alerted if the decoupling error exceeds the tolerance.

V. Decomposition

In this section we describe the algorithm for solving the nonlinear system of algebraic equations which models the

partitioned network at any given instant of time. Specifically, we show how the system of equations in Step 5 of Algorithm 3.1 is solved.

We begin by noting that the outputs of dormant subnetworks essentially appear as independent voltage and current sources feeding the inputs of adjacent alert subnetworks. The dormant model equations, thus, can be viewed as a part of the connection equations. Therefore, without loss of generality, we assume in the following development that all σ subnetworks are alert, and the system of equations which must be solved becomes

$$f_i(w_i, g_{k_i}(w_i), y_i, u_i) = 0, \qquad i = 1, \cdots, \sigma \quad (44)$$

$$\Theta_i y + u_i = \eta_i, \qquad i = 1, \cdots, \sigma \quad (45)$$

These equations are solved using the Newton–Raphson method which transforms the problem into one of solving a sequence of linearized equations. Each of these linearized systems is solved using the familiar LU factorization algorithm except that the block structure of the equations is used to advantage. We call the resultant scheme, Block LU Factorization.

5.1. Block LU Factorization

Let us combine the internal and output variables of the ith subnetwork into the vector $v_i = \text{col}\ (w_i, y_i)$. After discretization and linearization, the subnet equations can, therefore, be expressed as

$$J_i v_i + K_i u_i = r_i \quad (46)$$

where $J_i = \partial f_i / \partial v_i$ is the Jacobian of the ith subnetwork, $K_i = \partial f_i / \partial u_i$, and r_i is a right-hand side vector of the same dimension as v_i.

By appropriately padding the fan-in matrices Θ_i with zeros, we can also express the connection equations in terms of $v = \text{col}\ (v_1, v_2, \cdots, v_\sigma)$:

$$\Psi_i v + u_i = \eta_i. \quad (47)$$

Here, Ψ_i is the same as Θ_i augmented with enough zero columns to be conformable with v_i. Equation (47) can be further expanded by partitioning the matrix Ψ_i into σ submatrices Ψ_{ij} as follows:

$$\sum_{j=1}^{\sigma} \Psi_{ij} v_j + u_i = \eta_i. \quad (48)$$

Simply stated, Block LU factorization is a four-step process. The first step eliminates v_i from the subnet equations (46):

$$v_i = J_i^{-1}[r_i - K_i u_i] \quad (49)$$

by carrying out an LU factorization of each subnetwork Jacobian J_i followed by a forward substitution. The second step eliminates v_i from the connection equations (48) using the results of the first step:

$$u_i - \sum_{j=1}^{\sigma} \Psi_{ij} J_j^{-1} K_j u_j = \eta_i - \sum_{j=1}^{\sigma} \Psi_{ij} J_j^{-1} r_j \quad (50)$$

The third step solves the "reduced" set of linear connection equations (50), which are now expressed solely in terms of the inputs u_i. The final step is a back substitution on the subnet equations (49) which yields the values of v_i using the values found for u_i in the third step.

It is important to note that the Jacobian matrices J_i are not actually inverted as shown in (49) and (50). Rather their LU factors are calculated and then used in forward and back substitution steps as required.

The above Block LU factorization algorithm for solving the linearized network equations (46) and (48) has a number of interesting properties:

1) It is a direct solution algorithm, as opposed to a multi-level iterative algorithm [16], which obtains the values of the internal, output, and input variables "simultaneously."
2) The forward elimination and back substitution steps on the individual Jacobians can be carried out independently. Furthermore, these operations are independent of how the subnetworks are interconnected. This fact is used in the SAMSON2 program to automatically generate factorization and substitution code which contains a custom-made solution procedure for each different type of subnetwork.
3) If a subnetwork is dormant, the elimination of its internal and output variables from the subnet and connection equations reduces to simply substituting its dormant model y_i^D into the appropriate connection equations. Furthermore, the final back substitution step is not required. In effect, only the inputs to a dormant subnetwork are evaluated; they are needed to verify the dormancy assumption.

5.2. Block Gauss–Seidel Iteration

Equation (50) is a linear system of equations which must be solved for the inputs u_i so that the internal and output variables can be determined by back substitution on the subnet equations. The inputs u_i can be simply found by LU factorization. In the presence of dormant subnetworks, however, a more efficient scheme which can bypass the evaluation of input variables at "dormant" nodes would be more desirable. In this context, a node is any connection between two or more subnetworks. A node is considered dormant if all the subnetworks connected to it are dormant. The input variables corresponding to this node, therefore, appear to the rest of the alert subnetworks as though they were internal variables in a dormant "super" subnetwork and need not be calculated.

With this in mind, let us rewrite (50) on a node-by-node basis:

$$\begin{aligned} A_{11} z_1 + A_{12} z_2 + \cdots + A_{1m} z_m &= b_1 \\ \cdots\cdots\cdots & \\ A_{m1} z_1 + A_{m2} z_2 + \cdots + A_{mm} z_m &= b_m \end{aligned} \quad (51)$$

where $z_1, \cdots, z_m$ are vectors representing the input variables at nodes 1 through m, and $A_{11}, \cdots, A_{mm}$ are ma-

trices of the proper dimensions. Due to the strong connectivity of the variables at a given node relative to the connectivity of the variables of adjacent nodes, it seems reasonable to assume that the system of equations (51) is block-diagonally dominant, i.e.,

$$\|A_{ii}\| \geq \sum_{j=1, j\neq i}^{m} \|A_{ij}\|, \qquad i = 1, \cdots, m \tag{52}$$

where $\|\cdot\|$ is an appropriate matrix norm. The reduced connection equations, (51), can, therefore be solved, one node at a time, using a Block Gauss-Seidel iteration:

$$A_{ii}z_i^{k+1} = b_i - \sum_{j=1}^{i-1} A_{ij}z_j^{k+1} - \sum_{j=i+1}^{m} A_{ij}z_j^k,$$
$$i = 1, \cdots, m \tag{53}$$

where $k = 0, 1, \cdots$ is the iteration counter. Equation (53) is a small system of linear equations (the dimension of A_{ii} is equal to the number of subnetwork terminals connected to the ith node.) In SAMSON2, due to the manner in which the connection equations are formulated, the solution of (53) can be obtained trivially.

VI. Implementation: SAMSON2

The SAMSON2 system consists of a number of C programs which provide a user with the ability to specify a mixed circuit/logic level description of a VLSI network, define waveforms for its primary inputs, and request plots of a selected subset of its variables. The SAMSON2 software runs on a VAX-11/780 computer.

The two major components of SAMSON2 are the model compiler SAM1 and the event driven simulator SAM2. SAM1 is used to convert a textual description of a small circuit (such as a MOS device model or a logic gate) into a set of C subroutines which evaluate and solve the equations of that circuit [8]. Such a circuit is regarded as a *model* which can be instantiated in SAM2 to create *subnetworks*. For models described at the circuit level, the generated subroutines evaluate and solve a set of linearized algebraic equations. For models described at the logic level, the generated subroutines evaluate a set of 4-valued logic expressions using table lookup [19]. SAM2 performs automatic conversion between the continuous circuit waveforms and the discrete logic waveforms if adjacent subnetworks are modeled at different levels. Whereas the current implementation of SAMSON2 requires that the user manually partition a circuit before feeding it into SAM1 and SAM2, this error-prone chore is relegated in the next release of the software to a partitioning program called PRIMO [26].

Regardless of the modeling level, in a particular installation of SAMSON2, SAM1 is usually used to generate an object model library which is linked with the SAM2 kernel to produce an executable simulator image. In the remainder of this section we will describe the salient features of the circuit-level implementation in SAMSON2. Discussion of the logic-level algorithm and implementation in SAMSON2 will be the subject of a forthcoming paper.

6.1. *The Model Compiler SAM1*

SAM1 reads a textual description of a circuit-level model, sets up and orders its circuit equations, and finally writes out a set of C subroutines to evaluate and solve these equations. The language used to describe circuit-level models to SAM1 is similar in spirit to, albeit more powerful than, the SPICE2 input language. Its salient features include:

- free format input
- named model parameters
- node names (in addition to node numbers)
- arbitrary expressions to specify branch values
- interface to user-written nonlinear branch subroutines
- hierarchical instantiation of lower-level models.

The model equations are set up as a sparse tableau which is ordered to minimize fill-in and operation count during LU factorization. Variability typing [9] is used to distinguish among topological (± 1), constant, time-dependent, and variable entries in the tableau. The ordered tableau is then used to generate four C subroutines which load the tableau coefficients and perform LU factorization, forward substitution, and back substitution. The code to load the tableau and perform LU factorization is segregated into different sections based on the variability types of the entries. SAM1 also generates a data file which contains the names and default values of the model parameters, the names of the model variables, and the sizes of the various sparse arrays. This file is used by SAM2 to initialize each instance of the model.

6.2. *The Event Driven Simulator* SAM2

The SAM2 kernel includes a non-integer-time event scheduler, PBD integration and interpolation routines, and the control loop the EDCS algorithm. An executable image is produced by linking this kernel with a set of SAM1- and user-generated model subroutines. The separation of these model subroutines from the simulator core is one of the most powerful features of the SAMSON2 implementation alleviating the problem of "hard-wired" models, as is done in SPICE2.

The input of SAM2 consists of two parts: a net list and a command list. The net list specifies a set of interconnected subnetworks, each of which is an instance of a SAM1 model. The command list specifies the waveforms of the network inputs, the values of various control parameters (such as ϵ_a, ϵ_r and the length of the simulation), and plot requests of various variables. The waveforms of the requested variables are collected in a binary file which is fed into a graphical display program.

VII. Examples

We present in this section initial results for several MOS benchmark circuits to demonstrate the potential of the

TABLE I
SUMMARY OF SIMULATION RESULTS

Circuit	# MOSFETs	# Nodes	SPICE2 (s)	SAMSON2 LS (s)[a]	SAMSON2 ED (s)[b]	Ratio[c]
Inverter Chain						
# inv = 1	2	3	15.2	3.3	3.3	4.6
# inv = 10	20	12	536	93	64	8.4
# inv = 100	200	102	> 3 Hrs[d]	1733	425	28
31-Stage Ring Osc.	63	34	1307	235	106	12.3
RAM Array						
# bits = 1	6	7	71.7	3.1	3.1	23
# bits = 8	48	35	375.8	19.5	9.8	38
# bits = 128	768	515	5951	292	124	48
4 × 16 Decoder	120	41	678	88	64	11
16-Bit RAM + DEC	344	125	3799	548	189	20
Clock Generator	96	51	2446	827	794	3.1

[a] lock step mode
[b] event driven mode
[c] ratio of SPICE2 to event driven SAMSON2
[d] extrapolated result

event driven circuit simulation algorithm in the SAMSON2 program as compared to the traditional circuit simulation algorithm in the SPICE2 program. A summary of these results is shown in Table I. The simulation runs used to generate these results were made under the following conditions:

1) The MOS devices in all circuits were modeled by the Shichman–Hodges model [23], both in SAMSON2 and SPICE2.
2) The second order Trapeziodal integration method was used in SPICE2. For a fair comparison, the maximum order of PBD integration in SAMSON2 was set to 2.
3) The execution times indicated in Table I are in VAX-11/780 VMS CPU seconds, and account only for the transient analysis portion of the simulation. To eliminate any bias due to the manner in which the waveforms of requested variables are stored and displayed, these execution times were obtained by making special *timing* runs which did not request any waveforms to be produced.
4) In many implementations of SPICE2, there is a built-in restriction which limits the maximum size of integration steps to the user specified print interval. This restriction gives SAMSON2 an unfair advantage over SPICE2 especially during intervals with little activity in the circuit. To make the comparison more realistic, we made the print interval in SPICE2 equal to the total simulation period.
5) The relative error parameter was set to 0.1 percent in both programs. The setting of the absolute error parameters so that comparable accuracy can be obtained from SAMSON2 and SPICE2 proved to be quite tricky. Both programs use three absolute error parameters (for voltage, current, and charge), but they employ them in totally different ways. Therefore, we decided to optimize the choice of these parameters for each program separately, and to rely on a comparison of the generated waveforms to verify that they were generated with the same degree of accuracy.
6) For each circuit, two SAMSON2 execution times are given in Table I. These times correspond to the two modes in which SAMSON2 can be run. In lock step mode, all subnetworks are forced to take the same step size sequence. Thus except for partitioning, lock step mode is identical to traditional circuit simulation. In event driven mode, subnetworks are allowed to have separate step sizes and the event scheduler is used to control the simulation.

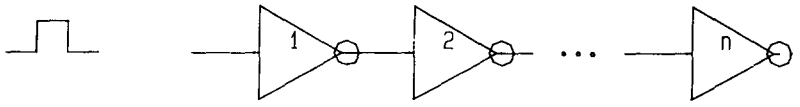
7.1. Inverter Chain

This circuit is a chain of identical MOS inverters varying in length from 1 to 100 (Fig. 2(a)). The circuit was partitioned so that each inverter formed a separate subnetwork. The chain was simulated for a period which was long enough to allow a 50 ns pulse supplied at the input of the first inverter to pass through the last inverter in the chain. Fig. 2(b) depicts the normalized CPU time (i.e., the CPU time divided by the simulation interval) [24] as a function of the chain length. This graph clearly shows that SAMSON2 is consistently faster than SPICE2 both in lock step and event driven modes. Furthermore, beyond 10 inverters in the chain, the effect of the temporal sparseness in the circuit begins to cause a significant reduction in the execution time for the event driven mode, and SAMSON2 becomes at least an order of magnitude faster than SPICE2. This effect can be dramatically illustrated with the *activity profile* for the 20-inverter chain shown in Fig. 2(c). In this figure, the activity of each subnetwork is represented by horizontal strip of vertical tick marks and dots. The tick marks denote the instants when the subnetwork was alert. The dots denote those times when the subnetwork had to be alerted because the dormancy test failed. The pattern in this figure clearly shows the input pulse traveling down the chain like a wavefront.

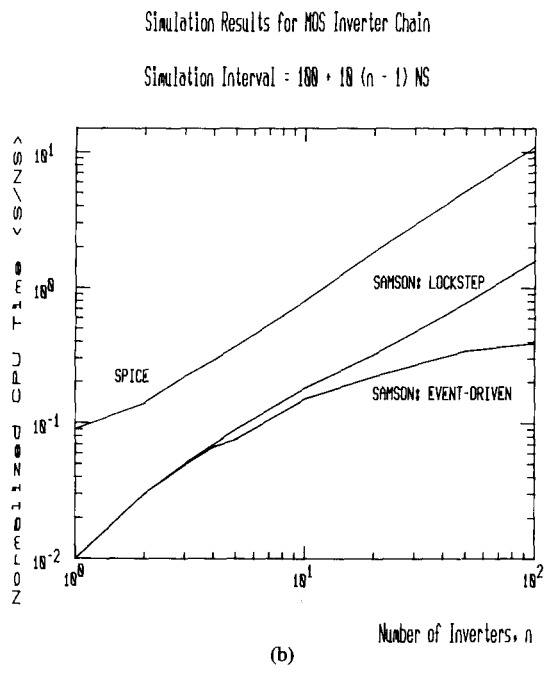
7.2. 31-Stage Ring Oscillator

This circuit, shown in Fig. 3(a) is one case where the event driven circuit simulation algorithm has a clear edge over waveform relaxation methods. As pointed out in [28], the convergence of waveform relaxation is slow for this circuit because of the feedback loop. One explanation for the slow convergence in this case is that waveform relaxation is noncausal since it is based on using future values of some waveforms to calculate the present values of other waveforms [15]. The scheme suggested in [28] to accelerate the convergence is to perform partial waveform relaxation on "windows" whose width is on the order of the propagation delay around the feedback loop. This enhancement of waveform relaxation, taken to the extreme of zero-width windows, approaches the event driven circuit simulation technique used in SAMSON2.

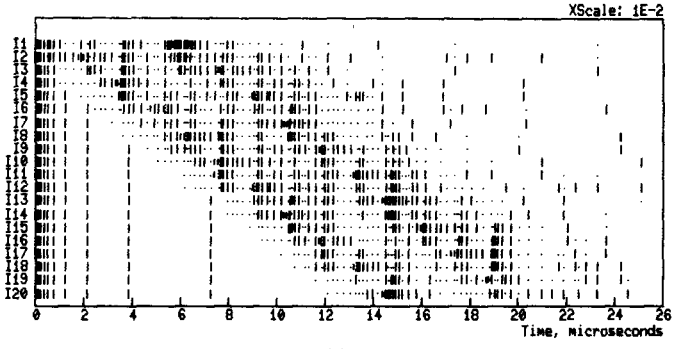
For the SAMSON2 simulation, the circuit was partitioned so that each of the 31 stages (1 NOR and 30 inverters) was a separate subnetwork. Fig. 3(b) and 3(c) show that the waveforms generated by SAMSON2 and SPICE2 for the node voltage of the last stage in the oscillator are in quite good agreement.



(a)



(b)



(c)

Fig. 2. Simulation results for MOS inverter chain. (a) Circuit diagram. (b) Execution time versus chain length. (c) Activity profile for 20-inverter chain.

7.3. Memory Circuits

This section describes the simulation results for memory circuits. Three experiments were performed. In the first, an n-bit array of RAM cells was simulated. In the second, a 4 × 16 decoder was simulated. And in the third, a 16-bit array of cells along with their 4 × 16 address decoder was simulated.

The RAM array, shown schematically in Fig. 4(a), is an n-bit array of MOS random access memory cells arranged as n rows of 1-bit words. The value of n was varied from 1 to 128 and a WRITE cycle to one row was simulated. The circuit was partitioned so that each cell was a separate subnetwork in SAMSON2. As shown in Table I, SAMSON2 exhibits the largest speedup over SPICE2 for this circuit because while 1 cell is being accessed, the remaining $n - 1$ cells show little activity. This can be clearly seen from the activity profile for the 8-cell RAM shown in Fig. 4(b).

The 4 × 16 decoder circuit is a standard NOR implementation of the decode function. The rather small speedup factor for this circuit is due mainly to its parallel nature. Specifically, since all 4 input lines and their complements fan out to the 16 4-input NOR gates, a change in a single input line will cause all 16 NOR gates to be scheduled for solution even though only two of them will actually flip their outputs.

7.4. 8-Phase Clock Generator

This circuit is the driver section of an 8-phase clock generator used to control a microprocessor. The circuit

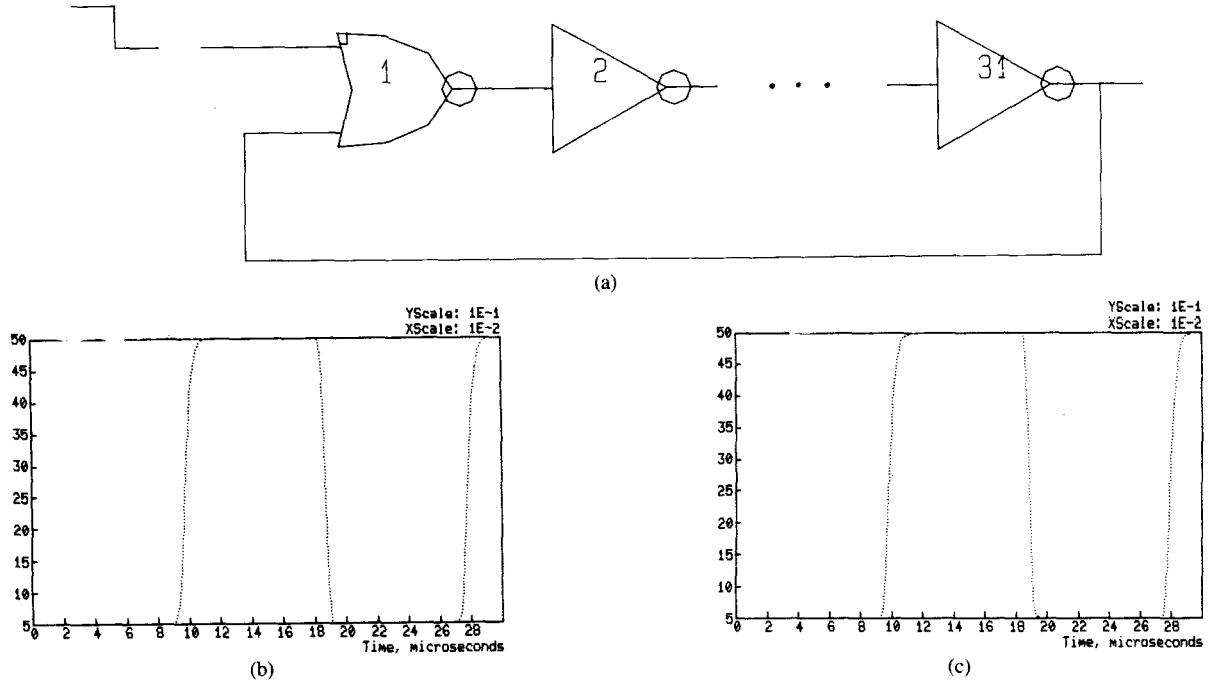


Fig. 3. Simulation results for 31-stage ring oscillator. (a) Circuit diagram. (b) Waveform generated by SAMSON2 for node voltage after last stage. (c) Waveform generated by SPICE2 for node voltage after last stage.

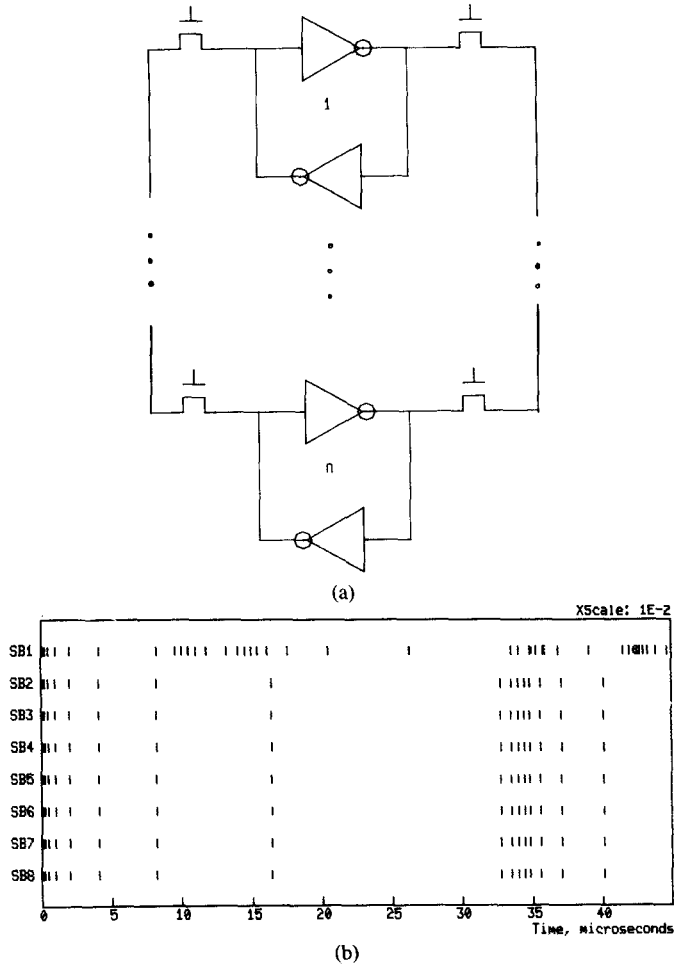


Fig. 4. Simulation results for n-bit RAM circuit. (a) Circuit Diagram. (b) Activity profile for 8-cell RAM.

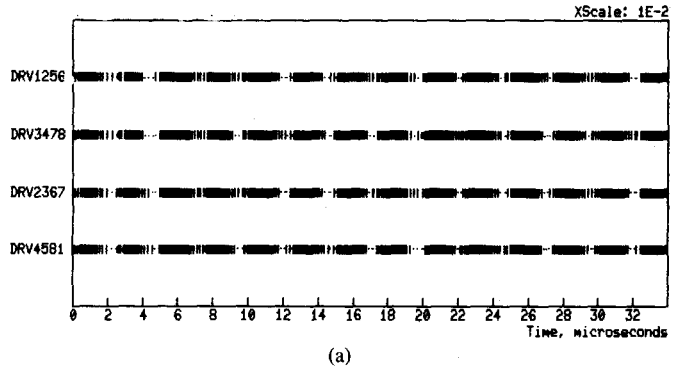
contains very large drivers, and has very tight loops with substantial feedback. Fig. 5(a) and 5(b) show the activity profiles obtained from event driven SAMSON2 simulations when the circuit is divided into four and sixteen subnetworks respectively. Both figures clearly demonstrate that this circuit is *temporally dense*. Nonetheless, for the four-subnetwork case SAMSON2 outperformed SPICE2 by a factor of 3. Fig. 5(c) shows the excellent agreement between SAMSON2 and SPICE2 for one of the node voltages.

VIII. Conclusions

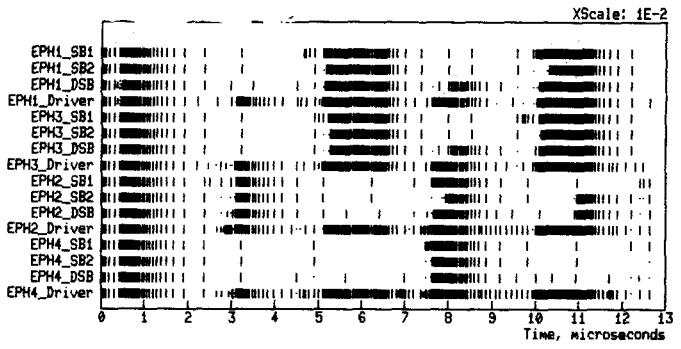
The EDCS algorithm provides a consistent mechanism for exploiting temporal sparseness in large VLSI circuits. Unlike earlier efforts in this regard, this algorithm is capable of estimating and controlling the errors inherent in the decoupling process and should, therefore, provide users with a greater degree of confidence in the simulation results.

Our limited initial experience with the SAMSON2 program on several actual MOS designs indicates that:

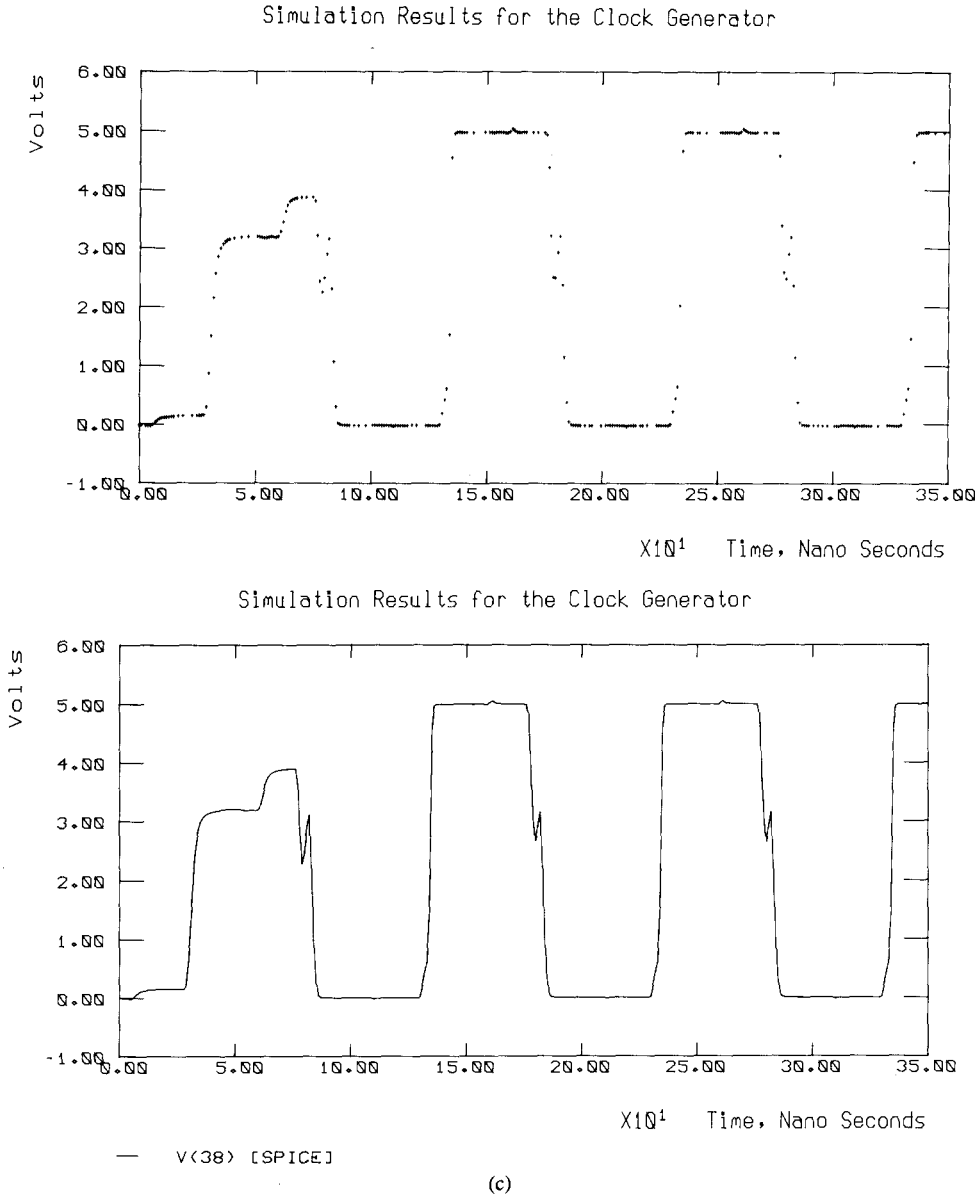
- there is little overhead in the event-scheduling process, even if the circuit lacks temporal sparseness
- the numerical integration and step-size selection scheme employed in SAMSON2 results in execution speeds which are at least 2 to 4 times faster than SPICE2
- for many large VLSI circuits, event scheduling techniques can result in execution time speedups of more than an order of magnitude over what is possible without such techniques.



(a)



(b)



(c)

Fig. 5. Simulation results for 8-phase clock generator. (a) Activity profile—four subnetworks. (b) Activity profile—sixteen subnetworks. (c) SAMSON2 and SPICE2 waveforms for a node voltage.

Finally, due to the event-driven basis of the algorithm, it is possible to realize a mixed circuit and logic simulation strategy. Specifically, it is possible to simulate a VLSI circuit in which some subnetworks, i.e., modules or cells, are characterized in terms of logical behavior, e.g., through Boolean expressions, and other subnetworks are characterized in terms of circuit level components, e.g., transistors and capacitors.

Having established EDCS as a viable method for fast and accurate circuit simulation, we still feel that further analysis of the algorithm is needed for a better understanding of its properties. Specifically, we would like to establish its stability characteristics and estimate its order of complexity.

Acknowledgment

The authors wish to acknowledge the efforts of Mahmud Buazza, Don Webber, and Steve Lammert in the development of SAMSON2 and the many thought-provoking discussions with Steve Greenberg.

References

[1] G. Arnout and H. J. DeMan, "The use of threshold functions and boolean-controlled network elements for macromodeling of LSI circuits," *IEEE J. Solid-State Circuits*, vol. SC-13, pp. 326-332, June 1978.

[2] R. K. Brayton, F. G. Gustavson, and G. D. Hachtel, "A new efficient algorithm for solving differential-algebraic systems using implicit backward differentiation formulas," *Proc. IEEE*, vol. 60, pp. 98-108, Jan. 1972.

[3] B. R. Chawla, H. K. Gummel, and P. Kozak, "MOTIS—An MOS timing simulator," *IEEE Trans. Circuits Syst.*, vol. CAS-22, pp. 901-910, Dec. 1975.

[4] G. Dahlquist and A. Bjorck, *Numerical Methods*. Englewood Cliffs, NJ: Prentice-Hall, 1974.

[5] S. P. Fan, M. Y. Hsueh, A. R. Newton, and D. O. Pederson, "MOTIS-C: A new circuit simulator for MOS LSI circuits," in *Proc. IEEE ISCAS*, pp. 700-703, 1977.

[6] C. W. Gear, "The automatic integration of stiff ordinary differential equations," in *Information Processing 68*, pp. 187-193, A. J. H. Morrell, Ed., Amsterdam, The Netherlands: North Holland, 1969.

[7] —, *Numerical Initial Value Problems in Ordinary Differential Equations*. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[8] D. J. Giannopoulos, "Implementation of a sparse matrix package for SAMSON," M.S. thesis, Dep. Elec. Eng., Carnegie-Mellon Univ., July 1981.

[9] G. D. Hachtel, R. K. Brayton, and F. G. Gustavson, "The sparse tableau approach to network analysis and design," *IEEE Trans. Circuit Theory*, vol. CT-18, pp. 101-113, Jan. 1971.

[10] C. Ho, A. E. Ruehli and P. A. Brennan, "The modified nodal approach to network analysis," *IEEE Trans. Circuits Syst.*, vol. CAS-22, pp. 504-509, June 1975.

[11] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli, "The waveform relaxation method for time-domain analysis of large scale integrated circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-1, pp. 131-145, July 1982.

[12] W. E. Milne, *Numerical Calculus*. Princeton, NJ: Princeton Univ., 1949.

[13] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Memo. ERL-M520, Electronics Research Lab., Univ. California, Berkeley, May 1975.

[14] A. R. Newton, "The simulation of large-scale integrated circuits," Mem. UCB/ERL M78/52, Electronics Research Lab., Univ. of California, Berkeley, July 1978.

[15] A. R. Newton, private communication.

[16] N. B. G. Rabbat, A. L. Sangiovanni-Vincentelli, and H. Y. Hsieh, "A multilevel Newton algorithm with macromodeling and latency for the analysis of large-scale nonlinear circuits in the time domain," *IEEE Trans. Circuits Syst.*, vol. CAS-26, pp. 733-741, Sept. 1979.

[17] D. J. Rose and R. A. Willoughby, "Sparse matrices and their applications," in *Proc. Conf. at IBM Research Center*, NY, Sept. 1971; New York: Plenum, 1972.

[18] K. A. Sakallah and S. W. Director, "An activity-directed circuit simulation algorithm," in *Proc. IEEE ICCC*, pp. 1032-1035, 1980.

[19] K. A. Sakallah, "Mixed simulation of electronic integrated circuits," Rep. DRC-02-07-81, Design Research Center, Carnegie-Mellon Univ. Nov. 1981.

[20] K. A. Sakallah and S. W. Director, "An event driven approach for mixed gate and circuit level simulation," in *Proc. IEEE ISCAS*, pp. 1194-1197, 1982.

[21] —, "SAMSON: An event driven VLSI circuit simulator," in *Proc. IEEE CICC*, pp. 226-231, 1984.

[22] R. A. Saleh, "Iterated timing analysis and SPLICE1," Memo. UCB/ERL M84/2, Electronics Research Laboratory, Univ. of California, Berkeley, January, 1984.

[23] Schichman, H. et. al., "Modeling and Simulation of Insulated Gate Field-effect Transistor Switching Circuits," *IEEE J. Solid-State Circuits*, SC-3:285-289, 1968.

[24] T. Trick. *Private Communication.*

[25] W. M. G. Van Bokhoven, "Linear Implicit Differentiation Formulas of Variable Step and Order," *IEEE Trans. Circuits and Systems*, CAS-22(2):109-115, February, 1975.

[26] A. V. Vasquez, "PRIMO User Guide," *to be published.*

[27] W. T. Weeks, A. J. Jiminez, G. W. Mahoney, D. Mehta, H. Qassemzadeh and T. R. Scott, "Algorithms for ASTAP—A network-Analysis Program," *IEEE Trans. Circuit Theory*, CT-20(6):628-634, November 1973.

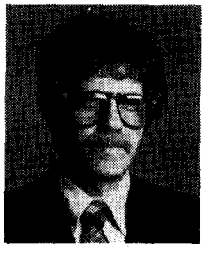
[28] J. White and A. L. Sangiovanni-Vincentelli, "RELAX2.1: A Waveform Relaxation Based Circuit Simulation Program," In *Proc. IEEE CICC*, pp. 232-236. IEEE, 1984.

*

Karem A. Sakallah (S'76-M'81) received the B.E. degree (with distinction) in electrical engineering from the American University of Beirut, Beirut, Lebanon, in 1975, and the M.S.E.E. and Ph.D. degrees in electrical and computer engineering from Carnegie-Mellon University, Pittsburgh, PA in 1977 and 1981, respectively.

In 1981 he joined the Department of Electrical Engineering at CMU as a Visiting Assistant Professor and since 1982 he has been with the Semiconductor Engineering Computer-Aided Design Group at Digital Equipment Corporation in Hudson, MA, where he is currently leader of the Analysis and Simulation Advanced Development team. His research interests are mainly in the area of computer-aided design of integrated circuits, with particular emphasis on numerical analysis, circuit and logic simulation algorithms, and timing verification of MOS VLSI circuits.

*



Stephen W. Director (S'65-M'69-SM'75-F'78) received the B.S. degree from the State University of New York at Stony Brook, Stony Brook, NY, in 1965 and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley, in 1967 and 1968, respectively.

In 1968 he joined the Department of Electrical Engineering of the University of Florida, Gainesville and was promoted to Professor in 1974. From September, 1974 to August, 1975 he was a Visiting Scientist in the Mathematical Sciences Department at IBM's T. J. Watson Research Center, Yorktown Heights, NY. He joined Carnegie-Mellon University as a Professor in 1977, where he is a Whitaker Professor and Head of the Department of Electrical and Computer Engineering, Professor of Computer Science, and Director of the Research Center for Computer-Aided Design. His research interests are in the area of computer-aided design and numerical methods and he is a consultant to industry in this area. He is also a Consulting Editor to McGraw-Hill Book Company. In 1981 he served as President of the IEEE Circuits and Systems Society and has also served as Secretary-Treasurer, President-Elect, and a member of the Administrative Committee of this Society. Further, he has been Chairman of the CAS Technical Committee on Computer-Aided Network Design (CANDE) and served as an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, as well as a guest editor of this Transactions and the PROCEEDINGS OF THE IEEE for special issues on computer-aided design. In addition to writing numerous technical articles, he has authored or coauthored three texts. In 1970 he received the Best Paper Award from the IEEE Circuits and Systems Society; in 1976 he received the Frederick Emmons Terman Award as the outstanding young electrical engineering educator from the American Society of Engineering Education; and in 1978 he received the W. R. G. Baker Prize Paper Award from the IEEE.

Dr. Director is a member of Sigma Xi and Eta Kappa Nu.