CIRCUIT ANALYSIS COMPUTING BY WAVEFORM RELAXATION

Conventional exact circuit simulation algorithms, as they are implemented in SPICE (1) and ASTAP (2) and in the follow-on programs, are limited by excessive compute time for the time domain analysis. The difference between the number of transistors that can be simulated and the number of transistors in a very-large-scale integrated (VLSI) circuit is an ever-increasing quantity. For large circuits, the compute time increases roughly as $O(n^{1.3})$ to $O(n^{1.8})$ depending on the circuit under analysis, where n is the number of circuit nodes. This has led to new approaches for the solution of these problems. The waveform-relaxation method (WR), which is such a technique, is an iterative approach for the exact solution of large VLSI circuits in the time domain.

The waveform relaxation technique, as it is presented in this article, aims at the same accuracy level as the widely used SPICE program (1). Often, however, in VLSI design, timing simulators are used to obtain solutions where accuracy is sacrificed for speed. These techniques are not considered in this article. However, three examples of such algorithms can be found in the references. The ITA algorithm (3) is based on time–point relaxation, whereas the SPECS algorithm (4) uses piecewise constant waveforms. Another technique that uses piecewise linear waveform approximations together with a highly damped explicit integration scheme is implemented in ACES (5).

A waveform method was first applied to VLSI circuit analysis problems in 1980. The starting point was the *one-way* circuit analysis formulation by Ruehli, Sangiovanni-Vincentelli, and Rabbat in 1980 (6), which ignored the gate-to-drain capacitive feedback in metal oxide semiconductor (MOS) transistors. Subsequently, the WR process for circuit simulation was invented by Lelarasmee, Ruehli, and Sangiovanni-Vincentelli to address this shortcoming (7). Different versions of WR-based circuit solvers were first developed at the University of California at Berkeley (8) and at IBM Research Laboratories (9) and later at several other locations. Numerous improvements and new applications have been discovered by the engineering and mathematical communities. On the mathematical aspects, Miekkala and Nevanlinna and Odeh (10) contributed much early on to the understanding of the convergence issues of WR.

Researchers now apply the WR approach to a wide range of problems from semiconductor device calculations (11), to nonlinear parabolic problems (12), and to multibody problems (13). In this article we will give only a limited set of references that highlight key advances in WR for both scalar and multiprocessor machines. A complete set of references up to 1986 are given in Ref. 14. A large chapter in Burrage's book (15) is dedicated to the application of WR to mostly *homogeneous* problems such as boundary value problems. It includes an extensive set of references on more recent WR work. The terminology *homogeneous* and *heterogeneous* is actually due to Gear. Homogeneous means problems that can be described by a single set of equations in which the domains have relatively uniform properties. The solution efficiency for homogeneous problems is a very strong function of the basic WR algorithm, which hereinafter will be referred to as the *internal* algorithm.

However, the focus of this article is on the heterogeneous VLSI circuit analysis problem. Heterogeneous problems consist of a multitude of different aspects such as linear and nonlinear parts. All these parts may have a mixture of different models embedded such as the conventional macromodels representing semiconductor devices. It is clear that a simple solution technique will be very inefficient for these problems. For the WR approach to be efficient the *internal* algorithms must be embedded in another layer, which we call the *external* algorithms.

The external WR solution algorithm can be characterized by the following steps:

1. Partitioning of a circuit into small subcircuits
2. Ordering of subcircuits
3. Scheduling of subcircuits for analysis
4. WR iteration until convergence of waveforms
5. Storing of waveforms in database

Before presenting the WR algorithms, it is appropriate to give some insight into the fundamental reasons why WR can be faster than a conventional time point circuit solver. Here, we assume that the circuits are sufficiently large that they can be partitioned into a reasonable number of subcircuits. First, in WR a large number of small matrices are solved rather than a single large one. For the usual modified nodal analysis circuit formulation (MNA) (16), the size of the matrix

is driven by the number of nodes in a circuit. The average solution time growth rate is $O(n^{1.5})$ for a circuit solver using sparse matrix techniques. It is obvious that the speedup due to matrix partitioning increases as the number of subcircuits increases, which is generally an increasing function of circuit size. This is obviously one of the factors why WR is fast for very large circuits. Also, each matrix can be solved using different time steps. The fact that the time steps in the subcircuits are different is called the *multirate* factor. The evidence that in large circuits the waveforms are most likely very different in different parts of a circuit points to the fact that this multirate behavior is another factor that increases strongly with the number of subcircuits. However, the speedup is reduced by the number of times the *average* subcircuit is evaluated due to WR iterations. Hence, it is obvious that the strategy is to keep the average number of WR iteration as small as possible. One of the factors that greatly helps is that a waveform solution error of 10^{-2} to 10^{-3} is sufficient for the circuit simulation problem. The typical number of WR iterations is between 3 and 4 for a well-partitioned circuit, while the WR iterations may vary between 2 and 20 for a typical heterogeneous circuit. Smaller errors, like those required for most homogeneous boundary-value problems, would demand a much larger number of WR iterations. For this type of problem, the convergence rate, considered later, is a much more important factor than for the circuit WR problem we consider here.

STRUCTURE OF VLSI CIRCUITS

Special-purpose solvers gain much of their efficiency from utilizing the specific structure of the problem at hand. VLSI circuit solvers are no exception. In fact, we hope that it will be clear from this section that a general-purpose WR solver without special partitioning algorithms would perform very poorly for VLSI circuits. We want to identify key properties of large VLSI circuits that make them good candidates for WR. Today's parallel computers make the analysis of circuits with less than several million transistors excellent candidates for WR.

As will be explained below, the partitioning step subdivides these very large circuits into small subcircuits, containing one to several hundred nodes. Figure 1 shows an example structure of a very large VLSI circuit. Each of the blocks may represent a functional unit of a VLSI chip with hundreds to thousands of transistors. It is immediately evident that these circuits should be partitioned into smaller

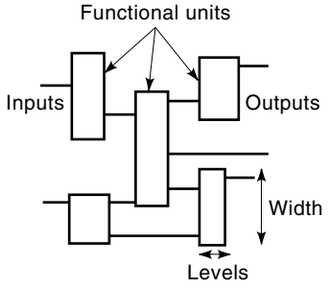

Figure 1. Basic structure of a large VLSI circuit as a set of blocks which are interconnected sparsely.

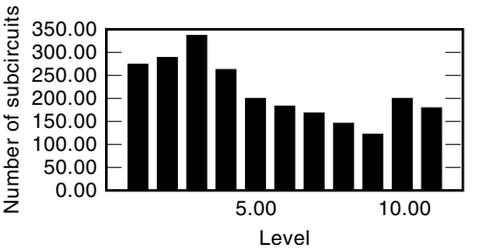

Figure 2. "Width" of a DRAM circuit as a function of the logic level, starting from the input.

units that may include one or more functional blocks depending on the block size. The connections between blocks shown in Fig. 1 may involve multiple paths. However, the external connections are usually sparse compared with the connections within the functional units. It is very important to recognize that the number of fanout connections of a circuit output is in general very sparse (e.g., 1–6). However, we have also encountered circuits with a fanout of 4000.

Each block has the property that the number of logic levels or the logical circuits that are connected in series must be limited to meet delay time limits or the system clock cycle. Hence, most functional units in Fig. 1 are relatively shallow in the number of levels. The functional units become "wider" as the number of transistors increases. An example is the error detection or correction circuitry of a 16 Mbyte dynamic random access memory (DRAM) design, shown in Fig. 2. The unit contains over 16×10^3 transistors; however, the number of logic levels is only 11. As can be seen from the figure, the "width" of the unit averages over 200 gates with a large potential for parallel processing. More insight into this will be given in the section entitled "Parallel Waveform-Relaxation-Based Circuit Simulation."

It is evident that the multirate factor increases rapidly as circuit size exceeds the size of a functional unit since the waveforms may have little correlation especially if they come from different functional units.

INTERNAL WR ALGORITHMS

In this section we examine the WR iteration process, assuming that a circuit has already been divided into subcircuits by the external partitioning algorithms considered in the section entitled "External Global WR Algorithms." The situation that we explore focuses on the *local* iteration between two neighboring subcircuits that are part of a large global circuit environment.

Fundamental WR Techniques

The waveform iteration process consists of an approximation to the solution of a set of nonlinear differential equations by a sequence of convergent waveforms. In the equations that follow, (w) is used to indicate the WR iteration index. It is assumed that subsystems or subcircuits are generated by the previously mentioned *external* partitioning and scheduling techniques. The *internal* algorithm is designed to solve subcircuit equations that are formed using the MNA approach as

$$C(x)\dot{x}(t) = g(x, t) \tag{1}$$

where $x = [v, i]^T$, v are node voltages, and i are selected currents. The nonlinearities in $C(x)$ are in part due to the transistor and integrated-circuit capacitances. To ensure that the solution is unique and that convergence for WR can be achieved, the capacitor and transistor models are designed with care so that they do not have discontinuities. The required properties of $C(x)$ and $g(x, t)$ are considered later in more detail in the section entitled "Convergence for the Nonlinear Case." We also do not want to consider the general differential algebraic equations (DAE) that result from general MNA equations since the resultant equations are more complex than the ordinary differential equations (ODE) case considered here, although it has been shown by several researchers that a solution is possible for the DAE case, for example, in Ref. 17.

Consider the scalar equation

$$\dot{x}(t) = f(x, t) \tag{2}$$

where $f(x, t) = C^{-1}g(x, t)$ with the initial condition $x(0) = x_0$, where $C > 0$ is a constant capacitor. We gain some insight into the waveform iterative solution by considering the Picard–Lindelöf (PL) iteration technique. In this method, the following waveform iteration is suggested:

$$\dot{x}^{(w+1)}(t) = f(t, x^{(w)}(t)) \tag{3}$$

$x^{(w+1)}(t_0) = x_0$, where x_0 is the initial value, which is the same for all iterations. It is assumed that we want to find the solution in a window in time $t \in [t_a, t_b]$, where t_a is the window start time and t_b is the window end time. For convenience we take the window to be $t \in [0, T]$, where T is the window size. In the PL technique, the solution of the problem is obtained by simply integrating the equation as

$$x^{(w+1)}(t) = x_0 + \int_0^t f(\tau, x^{(w)}(\tau))\, d\tau \tag{4}$$

where the initial waveform may be constant in the time window with $x^{(0)} = x_0$ and subsequent iterations yield new waveforms $x^{(1)}(t), x^{(2)}(t), x^{(3)}(t), \ldots$.

As an example, if the subsystem of equations is simplified by assuming that $g(x, t) = -Gx(t)$, where G represents a linear resistor R, or $G = 1/R$, then Eq. (1) is reduced to

$$\dot{x}(t) + \alpha x(t) = 0 \tag{5}$$

where $\alpha = 1/(RC)$ is the magnitude of the eigenvalue or inverse time constant. If we apply the PL iteration algorithm to this RC circuit problem we can make the following statement about the convergence of the iterative solution:

Theorem 1. If we apply the Picard–Lindelöf method to Eq. (5) on the interval $t \in [0, T]$, then the global error bound is given by

$$|x^*(t) - x^{(w)}(t)| \le \frac{(\alpha t)^{(w+1)}}{(w+1)!} \tag{6}$$

where $x^*(t)$ is the converged or the exact solution.

Proof: Applying the PL iteration Eq. (4) to Eq. (5) we can find the solution to be

$$x^{(w)}(t) = 1 - \alpha t + \cdots (-1)^w \frac{(\alpha t)^w}{w!} \tag{7}$$

This is the Taylor-series expansion for the solution $x^*(t) = e^{-\alpha t}$, where $\alpha = 1/RC$ and Eq. (6) is found from the error term in Taylor's theorem.

From this, we gain insight into the behavior of the solution of PL iterations for VLSI circuits that involve RC subcircuits with capacitances to ground. First, we observe what is called the *early-time* convergence property of the solution, which shows that it converges faster for small times. Also, the accuracy of the solution increases by one order for each iteration. By inspecting Eq. (6) we find that the time window $[0, T]$ must be kept small in relation to the number of waveform iterations such that $w \gg \alpha T$, to ensure uniform convergence. Hence, it may be desirable to subdivide the total analysis time into smaller subintervals or time windows for which convergence is obtained in fewer iterations. It is clear that the waveforms must be converged even at the end of the window at $t = T$ so that the previous solution will provide a good starting point for the next window. We will visit this question in more detail in the section entitled "Convergence for RC Circuits."

One-Way Systems and Gauss-Jacobi and Gauss-Seidel WR

For WR we assume that the system equation (1) has been split or partitioned according to the techniques described later in the section on partitioning. To study local convergence, we focus attention on the behavior between two connected subcircuits, and we temporarily ignore all interactions with other subcircuits. This is not representative of the real WR iteration scheme or schedule that involves all subcircuits. The local convergence situation is depicted in Fig. 3 where all other WR variables due to partitioning with respect to other subcircuits are assumed to be external (known) sources as shown. Hence, we assume that only the system variables x_1 and x_2 are relevant for the local convergence situation.

Local convergence of WR algorithms has been studied by many researchers [e.g., Lelarasmee, Ruehli, and Sangiovanni-Vincentelli (7), White and co-workers (18,19), and Debefve, Odeh, and Ruehli (14)]. We will look at the convergence issue in the next three sections. First, we give the most important features of the two main algorithms, the Gauss–Jacobi WR and Gauss–Seidel WR. They are best explained using the model in Fig. 3, where subcircuit 1 is excited by an input and coupling exists between the subcircuits in both directions, or

$$\begin{aligned} \dot{x}_1(t) &= f_1(x_1, x_2, u(t)) \\ \dot{x}_2(t) &= f_2(x_1, x_2) \end{aligned} \tag{8}$$

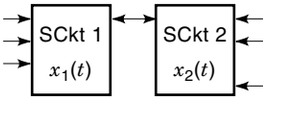

Figure 3. Two subcircuits shown to illustrate the local WR iteration process.

where $x_1(0) = x_{10}$ and $x_2(0) = x_{20}$ and $u(t)$ represents the inputs.

A special case exists if the connection from subcircuit 2 to subcircuit 1 is missing, or $\dot{x}_1(t) = f_1(x_1, u(t))$ only. In this case, we have a so-called *one-way* connection. If we solve the system by solving subcircuit 1 first, followed by subcircuit 2, the exact solution is obtained in one forward iteration (6). An example of such a system consists of two metal oxide semiconductor (MOS) transistor inverters without gate–drain feedback capacitances. Since most logic circuits are highly directional even during switching transients, it is evident that it is always advisable to solve the circuit in the direction of large coupling.

In the general case, with coupling in both directions, several iterations are necessary to obtain a solution. The Gauss–Jacobi WR iteration algorithm is given by

$$\begin{aligned} \dot{x}_1^{(w+1)}(t) &= f_1(x_1^{(w+1)}(t), x_2^{(w)}(t)) \\ \dot{x}_2^{(w+1)}(t) &= f_2(x_1^{(w)}(t), x_2^{(w+1)}(t)) \end{aligned} \tag{9}$$

where $x_1^{(w+1)}(t_0) = x_{10}$ and $x_2^{(w+1)}(t_0) = x_{20}$. The iteration sequence or schedule for this case is given by alternate evaluations of subcircuits 1 and 2, or 1, 2, 1, 2, . . . until convergence. In the Gauss–Jacobi (GJ) algorithm, all subcircuits are solved at iteration $(w + 1)$ using inputs from iteration (w). In contrast, the Gauss–Seidel (GS) WR method is given by

$$\begin{aligned} \dot{x}_1^{(w+1)}(t) &= f_1(x_1^{(w+1)}(t), x_2^{(w)}(t)) \\ \dot{x}_2^{(w+1)}(t) &= f_1(x_1^{(w+1)}(t), x_2^{(w+1)}(t)) \end{aligned} \tag{10}$$

where $x_1^{(w+1)}(t_0) = x_{10}$ and $x_2^{(w+1)}(t_0) = x_{20}$. In this approach, results that are computed in the solution of subcircuit 1 at iteration $(w + 1)$ are used in the solution of subcircuit 2 in the same iteration. This ordering and the immediate use of newly computed results allows the GS algorithm to take fewer iteration steps to converge than the GJ algorithm. For this reason, the GS method is generally preferred even though it puts a larger burden on the external WR algorithms such as ordering and scheduling, which have to select the subcircuit analysis sequence. It is not always possible to update all the variables as required for GS WR. For this case we will use what we call a *mostly GS* algorithm that instantaneously updates as many variables as possible. We will revisit this issue later in the section entitled "External Global WR Algorithms."

Successive Under- and Over-Relaxation WR

The idea of accelerating the solution by overestimating the update vector is used for most iteration techniques, including WR. The basic over-relaxation scheme (SOR) for GS WR takes a similar form as in the conventional scheme. A new set of waveform variables are introduced, which we call $y(t)$. With this the GS SOR WR scheme can be written as

$$\begin{aligned} \dot{y}_1^{(w+1)}(t) &= f_1(y_1^{(w+1)}(t), x_2^{(w)}(t)) \\ x_1^{(w+1)}(t) &= \beta y_1^{(w+1)}(t) + (1-\beta) y_1^{(w)}(t) \\ \dot{y}_2^{(w+1)}(t) &= f_2(x_1^{(w+1)}(t), y_2^{(w+1)}(t)) \\ x_2^{(w+1)}(t) &= \beta y_2^{(w+1)}(t) + (1-\beta) y_2^{(w)}(t) \end{aligned} \tag{11}$$

where $y_1^{(w+1)}(x_0) = x_{10}$ and $y_2^{(w+1)}(t_0) = x_{20}$. The over- or under-relaxation factor is usually in the range $0 \le \beta \le 2$.

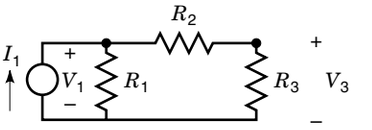

Figure 4. Resistive circuit to illustrate the iteration process if the circuit is partitioned at R_2.

The first practical application of SOR WR to VLSI circuit problems was done by Carlin and Vachoux (20). They applied under-relaxation to a stiff high-gain problem and showed that convergence could be improved by using $\beta < 1$. The definition of a *stiff* problem is one with a large difference in eigenvalues or time constants.

Convergence of WR SOR has been studied theoretically by Miekkala and Nevanlinna (21). It has also been applied to a semiconductor device problem by Reichelt, White, and Allen (22). They used a frequency-dependent over-relaxation factor $\beta(f)$ that was applied to the time domain through a convolution operator. The general time window under- and over-relaxation WR technique applied to VLSI circuit problems most likely can benefit greatly from a time-dependent factor $\beta(t)$ for $t \in [0, T]$.

Convergence for a Resistance Circuit

The convergence of the WR has been studied extensively for the linear circuits by several researchers, for example, Miekala and Nevanlinna (21) and Desai and Hajj (23). In this section, we look at the static case of the small resistance circuit, in Fig. 4, which is important for the partitioning step. The exact solution for this problem is given by

$$v_3 = I_1 R_1 \frac{R_3}{R_1 + R_2 + R_3} \tag{12}$$

which can be found by inspection.

For the iterative solution we define the forward gain $g_f = R_3/(R_2 + R_3)$ and the backward gain $g_r = R_1/(R_1 + R_2)$, which are simply the voltage divider ratios. This corresponds to splitting the circuit at R_2. The voltage dividers lead to the following voltage ratios: $v_3 = g_f v_1$ and $v_1 = g_r v_3$. The iterative solution yields

$$v_1^{(1)} = I_1 R_{p1},\ v_3^{(1)} = g_f v_1^{(1)},\ v_1^{(2)} = v_1^{(1)} + g_r v_3^{(1)} + \cdots \tag{13}$$

with $R_{p1} = (R_1 R_2)/(R_1 + R_2)$. With this, the iterative solution is given by

$$v_1^{(w)} = v_1^{(1)}[1 + g_f g_r + (g_f g_r)^2 + \cdots + (g_f g_r)^w] \tag{14}$$

The contraction factor is given by $\gamma = g_f g_r$. For convergence within a few iterations this factor needs to be $\gamma \ll 1$. Assume as an example that $R_1 = 1$, $R_2 = 10$, $R_3 = 5$. Then $g_f = \frac{1}{3}$ and $g_r = \frac{1}{11}$, which leads to $\gamma = \frac{1}{33}$. In this case convergence is reached in very few iterations to a very high accuracy. Also, directionality of coupling can be assigned, even with this simple circuit, as we observe since $g_f \gg g_r$. From the logic signal flow it is evident that in the directionality is assigned in the high-gain direction (14).

Convergence for *RC* Circuits

As was mentioned earlier, the convergence of WR for general circuits has been studied from the very beginning, and the impact of the capacitors on the convergence is considered a key issue. In the early work on WR it was assumed that each node in the circuit was required to have a capacitor to ground. More relaxed conditions have been established recently by Desai and Hajj (23) and Gristede, Ruehli, and Zukowski (24). Specifically, the convergence of *RC*-type circuits has been investigated by several researchers, for example, Miekkala, Nevanlinna, and Ruehli (25), Leimkuhler, Miekkala, and Nevanlinna (26), Ruehli and Zukowski (27), and Leimkuhler and Ruehli (28).

At first, it seems that many different *RC* circuit topologies need to be considered to gain an understanding of the WR behavior of *RC* circuits. However, in VLSI circuits there are two circuit topologies that appear many times as basic building blocks. The first involves a capacitor connected between two nodes, in which this capacitance may represent a gate-to-drain capacitance. To study its impact, the worst-case *RCR* situation is considered in Fig. 5(a), where only resistances are connected to the ground nodes. It should be noted that the usual sufficient conditions for WR convergence for example, Ref. 4 or 15, do not include this case.

It was shown in Ref. 25 that, even for the *RCR* circuit in Fig. 5(a), convergence can be achieved under certain conditions. The WR iteration equations for the case where we assume that a current source is connected to the left node in circuit in Fig. 5(a) are given by

$$\dot{v}_1^{(w+1)}(t) + \frac{1}{R_1 C_2} v_1^{(w+1)}(t) = \frac{I_1(t)}{C_2} + \dot{v}_3^{(w)}(t) \tag{15}$$

$$\dot{v}_3^{(w+1)}(t) + \frac{1}{R_3 C_2} v_3^{(w+1)}(t) = \dot{v}_1^{(w)}(t) \tag{16}$$

These local mapping functions show that the derivatives at one of the partitioned nodes v_1 or v_3 are a function of the derivative at the other end of the partition v_3 or v_1, respectively. It is intuitively obvious that for this case not only the input forcing functions but also the derivatives must be continuous for the WR iteration to converge. In actual VLSI circuits this issue is somewhat moderated since the gate-to-drain capacitances for the MOS field-effect transistors (MOSFET) have at least some capacitances to ground at each end. Again, the partitioning of the capacitance between gate and drain is very desirable in spite of the difficulties since a MOS transistor is a perfect one-way device if the capacitive coupling is ignored. This analysis emphasizes the requirements for the smoothness of the MOS capacitance models. It is confirmed by the mathematical analysis in Ref. 25 that slow WR convergence can be achieved for the limiting case in Fig. 5(a) in terms of Sobolev norms, which measure the derivatives as well as the functional values.

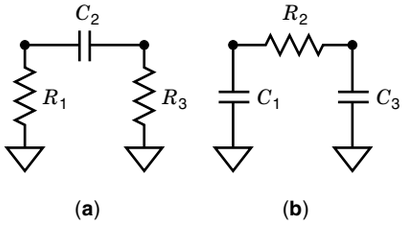

Figure 5. Two fundamental circuit topologies for partitioning as part of a VLSI circuit.

The second basic *RC* circuit is the low-pass *CRC* circuit, Fig. 5(b), which was analyzed by Ruehli and Zukowski (27) for simple cases, and for more complex *RC* circuits by Leimkuhler and Ruehli (28). Unlike the above *RCR* circuit, the *CRC* circuit seems to lend itself well to partitioning. However, there seems to be a problem in that it is hard to give a static argument for partitioning at the resistor R_2 as was done in the previous section for resistive circuits. The voltage transfer function for the case in which we excite the circuit with a voltage source V_s in series to C_1 is given by

$$\begin{aligned} \dot{v}_1^{(w+1)}(t) + \frac{1}{R_2 C_1} v_1^{(w+1)}(t) &= \frac{1}{R_2 C_1} v_3^{(w)}(t) + \dot{V}_s(t) \\ \dot{v}_3^{(w+1)}(t) + \frac{1}{R_2 C_3} v_3^{(w+1)}(t) &= \frac{1}{R_2 C_3} v_1^{(w)}(t) \end{aligned} \tag{17}$$

This can be written in the Laplace domain as

$$(s\boldsymbol{I} + \boldsymbol{M})\tilde{\boldsymbol{v}}^{(w+1)}(s) = \boldsymbol{N}\tilde{\boldsymbol{v}}^{(w)}(s) \tag{18}$$

where $\boldsymbol{M}$ and $\boldsymbol{N}$ are evident from Eqs. (17) and (18). We can rewrite Eq. (18) as

$$\tilde{\boldsymbol{v}}^{(w+1)}(s) = \boldsymbol{K}(s)\tilde{\boldsymbol{v}}^{(w)} \tag{19}$$

where the meaning of the symbol $\boldsymbol{K}(s)$ is evident from comparing the last two equations. The following theorem from Miekkala and Nevanlinna (10) is applied to the problem to find the spectral radius.

Theorem 2. Assume that the eigenvalues of $\boldsymbol{M}$ have positive real parts. Then the spectral radius of $\boldsymbol{K}(s)$ is $\rho(K) = \max_{\omega \in R}(j\omega\boldsymbol{I} + \boldsymbol{M})^{-1}\boldsymbol{N}$. For this case we have

$$\boldsymbol{K}(s) = \begin{bmatrix} 0 & \dfrac{1}{sR_2C_1 + 1} \\ \dfrac{1}{sR_2C_3 + 1} & 0 \end{bmatrix} \tag{20}$$

and it is clear that the minimum occurs for $s = 0$ where $\rho(K(0)) = 1$, which indicates that the convergence problem could occur at $s \to 0$.

In Refs. 27 and 28 it is shown that this problem does not occur for a finite time window. For convenience, we set both time constants to unity by choosing $C_1 = C_3 = R_2 = 1$. Then we can find the iterative solution to be

$$v_1^{(w)}(t) = e^{-t} \sum_{m=0}^{w-1} \frac{t^{2m}}{(2m)!} \tag{21}$$

With $(2m)! \simeq \sqrt{2\pi}(2m)^{2m+1/2}e^{-2m}$ the error term is

$$\text{Error}[v_1^{(w)}(t)] = \frac{e^{-t}}{2\sqrt{\pi}} \sum_{m=(w)}^{\infty} \left(\frac{\epsilon t}{2m}\right)^{2m} \frac{1}{\sqrt{m}} \tag{22}$$

From this we can derive the rapid convergence of the partitioned circuit provided that the window is small enough.

Theorem 3. The WR sequence converges rapidly in a time window T after the wth iteration for $w \leq \epsilon T/2$.

We can see from this that for our normalization $R_1C_2 = 1$ and $R_3C_2 = 1$ the convergence is very fast for $w \leq \epsilon T/2$. Hence, the larger the time constants of the two partitioned circuits, the larger the time window T for which rapid convergence occurs for a particular number of WR iterations w. It should be noted that this type of partitioning can again be done statically in the partitioning process by choosing an appropriate value of the time window T and the approximate number of WR iterations.

Another important observation can be made from this analysis on convergence behavior for windowing. First, most realistically modeled nodes for VLSI circuits, with the exception of the gate-to-drain capacitances, can be represented by the basic circuit in Fig. 5(b). Hence, the convergence behavior shown in this section given by Eq. (22) is quite typical. It shows that if the window T is chosen too large or equivalently, the number of WR iterations w are chosen to be insufficient, then the solution may be quite poor since the rapid convergence regime has not been reached. Specifically, at the window boundary $t = T$, the approximation and, even more important, the derivatives of the solution are approximated very poorly. Then, multistep integration techniques, such as the popular BDF2 method (29), that utilize solution points from the previous window are used to continue the solution in the next time window. This obviously represents a very poor starting condition for the solution in the next time window.

Convergence for the Nonlinear Case

Key aspects of VLSI circuits are the nonlinear MOSFETs and capacitances associated with the devices as well as the on-silicon diffusion wires and diodes. This requires a nonlinear analysis of the convergence, which has been available since the start of WR, for example, Lelarasmee, Ruehli, and Sangiovanni-Vincentelli (7), White et al. (18), White and Sangiovanni-Vincentelli (19), and Debefve, Odeh, and Ruehli (14). However, the nonlinear WR convergence proofs have become more general in recent years. The proofs by Schneider (17) and Gristede, Ruehli, and Zukowski (24) take the DAE (differential-algebraic equation) for the MNA circuit formulation (17) into account. Also, more useful bounds have been derived with a one-sided Lipschitz constant by in 't Hout (30) and in Burrage's book (15). Here, we give an interesting and relevant proof of Taubert and Wiedl (31) that illuminates the nonlinear convergence in terms of a time window T. The vector u is given by $u = (u_1, u_2, \ldots, u_m)$, and the two relevant norms are $\|u\|_\infty = \max_{i=1,\ldots,m} |u_i|$ and $\|z\|_T = \max_{t\in[0,T]} \|z(t)\|_\infty$.

The circuit equations for a MOSFET circuit including the voltage-dependent capacitors are given as [similar to Eq. (1)]

$$C(x)\dot{x}(t) = G(x,t) \tag{23}$$

with the initial value $x(0) = x_0$. All the conditions below are assumed to apply in a window in time, which we choose to be $t \in [0,T]$. The voltage excursion must be contained for the semiconductor devices such that the nonlinearities can be described by a valid circuit model. Hence, we assume that limits are also applied on the particular values of x so that the conditions of the theorem are met.

First, we assume that the transistor nonlinearities satisfy the Lipschitz continuity condition

$$\|G(x_1,t) - G(x_2,t)\|_\infty \leq K\|x_1 - x_2\|_\infty \tag{24}$$

for the allowed values of t, x_1, x_2. Second, the nonlinear behavior of the $m \times m$ capacitance matrix with respect to the real vector z of length m must satisfy several conditions. Each element of the capacitance matrix $C(z)$ must satisfy another Lipschitz continuity condition with a constant L that applies for all j for the real vectors u, v:

$$\sum_{k=1}^{m} |c_{jk}(u) - c_{jk}(v)| \leq L\|u - v\|_\infty \tag{25}$$

A further condition is imposed on the capacitances. We assume that there exists a constant $\alpha > 0$ such that for all real vectors, u, v where $u > 0$ we have

$$[C(z)u]_i u_i \geq \alpha\|u\|_\infty^2 \tag{26}$$

This condition can be viewed as being related to the instantaneous energy in the system of capacitances C, which is given by $1/2u^TCu > 0$ for $u \neq 0$. For a nodal capacitance matrix, this implies diagonal dominance. For the nonlinear case, the requirements in Eq. (26) are somewhat more restrictive than what is required for the multiple capacitances for which $z = u$.

The WR equations for two subcircuits in which each subcircuit is represented by a single equation are given by

$$\begin{aligned} c_{11}(x_1^{(w+1)}, x_2^{(w)})\dot{x}_1^{(w+1)}(t) + c_{12}(x_1^{(w+1)}, x_2^{(w)})\dot{x}_2^{(w+1)}(t) \\ = G_1(x_1^{(w+1)}(t), x_2^{(w)}(t), t) \\ c_{21}(x_1^{(w+1)}, x_2^{(w+1)})\dot{x}_1^{(w+1)}(t) + c_{22}(x_1^{(w+1)}, x_2^{(w+1)})\dot{x}_2^{(w+1)}(t) \\ = G_2(x_1^{(w+1)}(t), x_2^{(w+1)}(t), t) \end{aligned} \tag{27}$$

where the contribution of the nonlinear capacitances is evident. The initial conditions are $x_1^{(w+1)}(t_0) = x_{10}$ and $x_2^{(w+1)}(t_0) = x_{20}$. Note that in terms of capacitances $c_{11} = C_1 + C_2$, $c_{12} = c_{21} = -C_2$, and $c_{22} = C_2 + C_3$, where the circuit consists of three capacitances with the same topology as the circuits in Fig. 5. All three capacitances C_1, C_2, and C_3 can be nonlinear.

Now, we are ready to state the very interesting condition for nonlinear convergence in a time window $[0,T]$.

Theorem 4. The sequence of approximate solutions given by the WR iterations $x^{(w+1)}$ converges uniformly to the solution x^* of Eq. (23) in $[0,T]$ for which the following condition holds:

$$\alpha - KT - LT\|\dot{x}^*\|_T \geq 0$$

Proof. Here, we only give an outline of the proof. The uniqueness of the solution of Eq. (23) is guaranteed by the conditions given earlier in a time window $t \in [0,T]$. For any continuous differentiable function $x(t)$ with the initial condition $x(0) = x^*(0)$, we form the difference

$$A(x,x^*) = C(x)\dot{x} - C(x^*)\dot{x}^* - G(x,t) + G(x^*,t) \tag{28}$$

which can be expanded by adding and subtracting the quantity $C(x)\dot{x}^*$. For $\hat{t} \in [0,T]$ we form the quantity

$$A(x, x^*)_i(\hat{t})(\dot{x} - \dot{x}^*)_i(\hat{t}) \tag{29}$$

Using also the fact that

$$x_i^{(w)}(t) = x_{i0} + \int_0^t \dot{x}_i^{(w)}(s)\, ds \tag{30}$$

and using the Lipschitz conditions in the expanded form of Eq. (29) with K, L from above, we can show the inequality in Theorem 4. We assume that the size of the time windows T is adjusted during the transient analysis.

It is evident from this that both the nonlinearities of the capacitances and the devices can reduce the maximum size of the time window during the highly nonlinear transitions of the devices for which usually the smallest time windows T occur. This transition time is usually a small part of the transient analysis time. Also, the transition is the time where the circuit solver will take very small time steps, so that T includes a reasonable number of time steps.

Newton Variant of WR

Given Eq. (2), in a general form, the WR schemes considered so far first partition the system at the differential equation level. Then the nonlinear equations are solved separately for each subcircuit using Newton's method. Van Bokhoven (32) considered a variation on WR by essentially interchanging the waveform loop with the Newton linearization of the equations. Hence, the Newton variant of WR starts by linearizing Eq. (2) for the entire circuit. The system of equations rewritten in a functional form is

$$F(x) = C(x,t)\dot{x}(t) - g(x,t) = 0 \tag{31}$$

This form can be linearized using the Newton scheme as

$$x^{(n+1)} = x^{(n)} - J_{\mathrm{F}}^{-1}(x^{(n)})F(x^{(n)}) \tag{32}$$

where $J_F(x)$ is the Fréchet derivative of F and where n is the nonlinear or Newton iteration index. This method has been explored by many researchers [e.g., (18,19,32–34)]. It can be shown that the resultant scheme

$$\dot{x}^{(n+1)} - J_n x^{(n+1)} = f(x^{(n)}) - J_n x^{(n)} \tag{33}$$

is another splitting of the circuit matrix of the entire circuit (15). If we apply only a single Newton iteration $n = 1$, we can partition the resultant circuit matrix and we can use an external waveform relaxation loop. At this level, all the necessary algorithms such as windowing are applied.

The Newton waveform technique has been successfully applied to the homogeneous semiconductor problems by Lumsdaine and White (11). These problems do not require a complex partitioning procedure as is the case for heterogeneous systems. For homogeneous problems the Newton waveform approach is preferred for its quadratic convergence behavior. However, this is more of an issue for the solution of homogeneous systems since much more accuracy and therefore a much larger number of WR iterations are required than for the circuit simulation problem.

Other issues are of importance for a large VLSI circuit problem for which the general WR algorithm offers several advantages over the Newton waveform approach. First, for the conventional WR, circuits are partitioned at the schematic level into self-contained subcircuits that are analyzed independently using a conventional circuit solver. Furthermore, the interaction between subcircuits and functional units, at all levels, simply consists of the exchange of segments of waveforms of various sizes. Other techniques such as the hierarchical WR techniques (35) and parallel WR discussed later benefit greatly from the simplicity of the conventional WR approach.

EXTERNAL GLOBAL WR ALGORITHMS

In this section we consider the overall environment that is required for a heterogeneous VLSI circuit WR solver where the circuit structure may be extremely nonuniform. This is especially true for mixed analog-digital circuits. To make the issue more complex, feedback loops in logical circuits may require more WR iterations than the local WR interfaces require to converge. It is evident from this that all aspects of a WR program must be implemented carefully to obtain maximum overall efficiency. Furthermore, it is very unlikely that the optimal number of WR iterations is uniformly the same for all *local* interfaces between the subcircuits. Before we can consider these *global* convergence issues at the end of this section, we first must introduce other fundamental concepts such as partitioning, ordering, and scheduling. A detailed description of the concepts is given in Ref. 14. A key aspect of the external environment is the storage of the waveforms. As will be evident below, the waveforms for iterations w and $w + 1$ must be available for computations.

Partitioning

The partitioning of a circuit into small subcircuits is clearly a heuristic process for heterogeneous systems. One of the key driving factors for partitioning is that convergence of the internal WR algorithms must be enhanced by the partitioning process. This has been recognized since the beginning of VLSI WR, and much work has been dedicated to this issue throughout the evolution of WR [e.g., Carlin and Vachoux (36) White and Sangiovanni-Vincentelli (37), Cockerill et al. (38), and John, Rissiek, and Paap (39).

Definition 1. *Partitioning* means subdividing a large circuit (Ckt) into small subcircuits (SCkts). The SCkts are chosen in such a way that coupling between subcircuits is minimized and that convergence is enhanced.

It should be noted that this type of partitioning is also known as multisplitting or diacoptics. Most partitioning algorithms are static; the partitions are defined before a transient analysis is performed. In fact, it is the first step in the overall WR scheme. Some exploratory work on dynamic partitioning has been done by Dumlugol, Cockx, and DeMan (40) for specific circuit structures in which the partitions are altered during the iteration process. It is evident that for large heteroge-

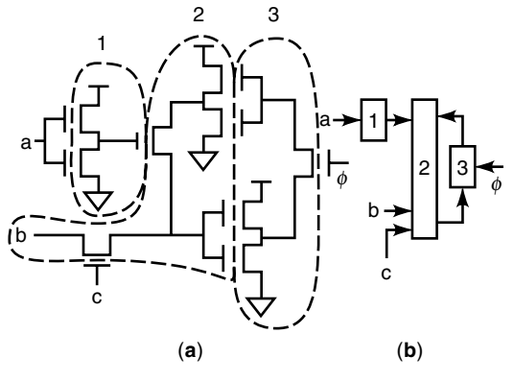

Figure 6. (a) A MOS transistor circuit partitioned into three subcircuits; (b) A directed graph corresponding to Fig. 6(a), which shows the main logic signal flow.

neous circuits static partitioning is preferred since it can be designed for all types of structures.

Two of the most popular methods are *pointwise* and *block* partitioning. Pointwise partitioning breaks the circuit at each node, generating subcircuits with one node each. This scheme does not control the coupling between subcircuits. On the other hand, block partitioning groups one or more nodes into SCkt based upon estimates of the coupling of the circuit elements that connect between them. The techniques in the previous sections entitled "Convergence for a Resistance Circuit" and "Convergence for *RC* Circuits" are applied to see if two nodes should be in the same SCkt by evaluating the potential coupling. The nodes of the resultant SCkt are then ensured not to be strongly coupled to other SCkts at least in one direction. This direction is away from the SCkt for an output and into the SCkt for an input. Hence, block or subcircuit partitioning leads to much faster WR convergence.

Most WR programs also use graph theoretic partitioning algorithms like the *strongly connected* or *dc connected* components (14). An example of a circuit that has been partitioned into dc connected components is shown in Fig. 6(a). In Fig. 6(b), a directed graph is shown that corresponds to the circuit in Fig. 6(a).

Next, we consider in more detail the decision process for the assembly of nodes into SCkts. One of the algorithms used is the diagonally dominant Norton (DDN) algorithm by White and Sangiovanni-Vincentelli (37), which is based on techniques given in the section on resistance-circuit convergence for static partitioning. This algorithm is based on the idea that two nodes may either be coupled only resistively or capacitively. The simple circuit in Fig. 4 provides a model for applying this algorithm. Consider the resistor R_2 to represent all parallel conducting paths between any two nodes. These include all resistive and inductive elements as well as "worst-case" values for the nonlinear conductances of semiconductor devices. The inductance voltage drops are set to zero for the conductance between nodes. The resistances R_1 and R_3 represent the equivalent resistance of all local paths to ground. Again, this is done by ignoring all capacitance in the circuit for these two nodes. If the convergence factor $\gamma = g_f g_r$ is greater than some threshold value, usually chosen to be between 0.3 and 0.95, the two nodes are considered to be strongly coupled and are placed into the same subcircuit. Since the model is set up using the worst case for nonlinear resistances, a slightly higher value of the threshold is used since the gain estimates are conservative. It should be noted that the same technique can be used for a circuit that includes only capacitors in exactly the same way as in the section on convergence for *RC* circuits in which the "equivalent" resistance values used are given by $R = 1/C$. All pairs of nodes that are directly connected to one another are considered in the partitioning process, and the algorithms just described will decide whether to place them in the same subcircuit or not. Hopefully, the resultant SCkts are small so that each has only a few nodes. However, if too many single-node subcircuits result, it may be advantageous to merge some subcircuits into larger ones. Merging or condensing will reduce the number of WR iterations at the expense of having to solve larger subcircuits. An example of a situation where it may be advisable to condense subcircuits occurs when global feedback loops exist in the circuit. This will be explained in more detail in the next section.

Ordering and Scheduling

Definition 2. *Ordering* is defined as the process of labeling the subcircuits in an increasing order starting with the one(s) that should be solved first.

It is evident from the example of a *one-way* inverter chain, Fig. 7, in which the gate-drain capacitances are ignored, that the solution starting from input to output leads to convergence in one WR iteration. On the other hand, if the subcircuits are ordered from output to input, m inverters require m WR iterations for convergence. It was shown in the first section that large VLSI circuits that are simulated by WR techniques have many parallel paths, which can result in the same ordering in each of the paths or chains of SCkts.

Ordering becomes more difficult for circuits with feedback loops. There are two possible choices for dealing with feedback. An example is given in Fig. 6(a) for a circuit with feedback. This is apparent from the graph in Fig. 6(b). For small feedback loops involving only a few SCkts like that given in this example, it may be more efficient to form a larger SCkt by merging all the SCkts in a feedback loop into a single SCkt. For larger loops, it may be better to cut the feedback loops (14). Application of these techniques results in a new set of SCkts without cycles. The feedback-loop-cutting algorithm can also be viewed as a *mostly* Gauss–Seidel approach in which the values at the feedback input nodes are specified at iteration $(w + 1)$ by using feedback values from $x^{(w)}$ as is done for all variables in the Gauss–Jacobi technique. All the other nonfeedback variables are updated in the GS fashion. So-called *strongly connected component* techniques (41) are used to detect the inputs to the feedback loops, which are the

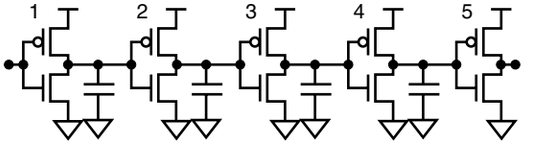

Figure 7. Chain of MOSFET inverters which is used to illustrate different scheduling techniques.

GJ variables. An ordering for the resultant circuit in terms of the SCkts is found by the leveling algorithm.

Leveling Algorithm.

Input: SCkt graph

Output: Assignment of Ckts to ordering levels

```
Function LevelizeSubCircuits
{
  LevelNumber = 0;
  Assign inputs to LevelNumber = 0;
    REPEAT
      FOR each SCkt s in LevelNumber {
        FOR each SCkt k in fanout set for s {
          NumberOfOrderedInputs =
            NumberOfOrderedInputs + 1;
            IF NumberOfOrderedInputs ==
              NumberOfInputs
              assign SCkt k to LevelNumber + 1;
        }
      }
      LevelNumber = LevelNumber + 1;
  UNTIL Level with LevelNumber is empty;
}
```

For scalar WR it is sufficient to order the SCkts in such a way that the transient analysis of each SCkt in a lower level is scheduled before the next higher-level SCkt is so that all the input variables at iteration $(w+1)$ are available before the transient analysis for a SCkt is performed.

Definition 3. *Scheduling* means the *scheduling for analysis* of a subcircuit according to the ordering until WR convergence is achieved.

Most existing WR-based circuit solvers use what is called *basic* scheduling. However, other scheduling methods may be more efficient than this approach. Here, we consider a simple chain of ordered SCkts to illustrate the different techniques. To make the example applicable to all scheduling techniques of interest, we use a special circuit for which we can overlap some of the nodes of any pair of SCkts. The overlap means that a subset of nodes in a subcircuit may be shared between two neighboring subcircuits. We take the chain of five inverters in Fig. 7 as a simplified example and order it from input to output as is shown in Fig. 7:

1 2 3 4 5

For a basic schedule, the SCkts are analyzed according to the ordering starting from the input, and a basic analysis schedule is given by

1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

where we assume in this example that global convergence has been reached in three WR iterations. If there are several time windows in the analysis, we execute this schedule for each of the windows separately. It is well known that the number of WR iteration for convergence is very nonuniform for the different time windows due to the switching activities of the transistors.

To properly explain the overlap scheduling technique, we assume that each of the subcircuits has several nodes unlike the very simple inverter chain in Fig. 7, which has only one node per subcircuit. It is intuitively obvious that if we could overlap or share some of the nodes of the neighboring SCkts during the local WR iteration, then the convergence would be enhanced at the cost of having to analyze the shared nodes twice as many times at each iteration. Here we assume that each SCkt now has three nodes instead of one. We still assume that we have a chain a five SCkts, where the labels of the three internal nodes per SCkt are given as

11 12 13 21 22 23 31 32 33 41 42 43 51 52 53

It is evident that many different overlaps can be chosen in this example even if only the boundary nodes are shared between subcircuits. To illustrate the fact that the overlap does not even have to be symmetrical we give the following overlap example:

11 12 13 21 21 22 23 31 31 32 33 41 41 42 43
51 51 52 53
11 12 13 21 21 22 23 31 31 32 33 41 41 42 43
51 51 52 53

Specifically, we chose an overlap in which one node of the next SCkt is taken into account while analyzing a SCkt. However, we assume that it is a waste of compute time to also take the corresponding node from the previous subcircuit into account. An example for symmetric overlap for subcircuit 2 would be 13 21 22 23 31. From this it is evident that the number of nodes in each SCkt analysis can grow rapidly with overlap scheduling. Hence, the reduced number of WR iterations must be balanced against the analysis of larger subcircuits. We observed experimentally that overlap scheduling does not work well for circuits in which the coupling is sufficiently weak as is the case for an inverter chain. Its main application is for situations in which the coupling is strong for a large number of nodes such that large subcircuits result. Overlap scheduling works well even if the circuits are very strongly coupled. Hence, it is best applied to severe coupling situations. Overlap scheduling has been applied in different forms since the beginning of WR. The first paper applying overlap scheduling was by Mokari-Bolhassan, Smart, and Trick (42), and a thorough mathematical analysis and further extension were given by Jeltsch and Pohl (43). For heterogeneous circuits overlap scheduling is especially applicable to the situation in which the coupling is very strong such that some very large subcircuits would result. In this situation, the additional cost of the overlap is offset by other gains in compute time.

Another important method with the potential to improve the overall efficiency of WR is ϵ scheduling by Odeh, Ruehli, and Carlin (44). To show how the method works we consider the coupling in a matrix system rather than a system of differential equations. A typical form for the systems is

$$(L+E)x = b \tag{34}$$

where L is a lower triangular invertible matrix representing strong forward coupling and E is a matrix with a sparse array of small coupling terms of $O(\epsilon)$ and zeros in all other locations. For simplicity we assume that the small couplings in E can be arranged in a vector, which we again call $E = (v_1, v_2, v_3, \ldots)$ to retain meaning. For a given m we divide the vector into two parts, $E = E_1 + E_2$, where $E_1 = (v_1, v_2, v_3, \ldots, v_{m-2})$ and where E_2 is the remainder of the vector. We need to note that the feedback element v_s corresponds to the variable X_{x+1}. With this we can see that if we ignore the feedback from the variables x_j, $j > m$ one simply has to set all elements in E_2 to zero. We denote by y the variables of the truncated system corresponding to Eq. (34); then the new system is

$$(L + E_1)y = b \tag{35}$$

We now can define the error vector $e = x - y$, which is due to the truncation of the $O(\epsilon)$ feedback variables. The following will give an indication of how the errors propagate in both the forward and backward directions.

Theorem 5. For a system of size N, the components of the error e for the truncated system, Eq. (35), in comparison with the fully coupled system, Eq. (34), is given by $e_k = O(\epsilon^{N-k})$ for the backward direction $0 \le k \le m - 1$. The error in the forward direction is given by $e_k = O(\epsilon)$ for $k \ge m$.

The proof of this theorem is given in Ref. 14. It gives us a clear indication of how the scheduling can be changed to improve global convergence. One key observation is that the errors decay rapidly in the back direction due to the ϵ backcoupling. This implies that a very good result can be obtained for the present SCkt even if we have not analyzed all the SCkts with a higher order. On the other hand, once an error has been committed somewhere along the chain it will propagate forward to all SCkts with a higher order until the error has been corrected with further iterations.

We can utilize this result to come up with a scheduling that we call ϵ scheduling, for now-obvious reasons. The ϵ schedule is applied locally and it "propagates" forward, making sure that convergence is achieved locally after all the WR iterations have been executed. For the inverter chain example an ϵ schedule is given by

```
1 2
1 2 3
  2 3 4
    3 4 5
      4 5
```

Techniques such as overlap scheduling and ϵ scheduling clearly are more difficult to apply for complex circuits with complicated fanout situations due to the complicated partitioned circuits and logical signal flow. We give results for an inverter chain, which is the simplest circuit with which to illustrate these concepts. We consider chains with 16, 32, and 64 inverters that are partitioned into SCkts where each SCkt has two inverters. This partition was found to give the best results in all cases. Table 1 compares a conventional circuit analysis result with a basic and ϵ-schedule WR analysis. These results give insight into the general behavior of the solution gain for WR over conventional circuit simulation. First we observe that very small circuits have little multirate and matrix overhead, so one would expect the conventional SPICE type solution to be faster than WR, which is indeed the case. The other interesting comparison is between different scheduling algorithms for which the difference is a nonmonotonic function. We did not try to apply overlap scheduling to this since the coupling for the inverter chain is moderate. It would not be a fair test for overlap scheduling, which excels in strongly coupled situations, with a weakly coupled example. Finally, we would like to point out that the scheduling techniques can be combined. For example, overlap and ϵ scheduling can be used for different parts of the same circuit.

Table 1. Conventional Circuit Analysis *vs.* WR Analysis

Ckt Name	No. Trans.	Time (s)		
		Conventional	Basic Sched. WR	ε-sched. WR
Ch8	16	26	41	34
Ch16	32	118	100	48
Ch32	64	635	270	171

Global Convergence

With the techniques described previously we are ready to consider the difficult global convergence issues. The local convergence between two SCkts has been examined extensively earlier in the section entitled "Internal WR Algorithms." Global convergence deals with the interaction of thousands of SCkts with different waveform interfaces. The gain or loss of efficiency due to the global algorithms can be considerable. A single local waveform interface with poor partitioning can slow down the convergence of a large circuit. The well-known example is the one-way inverter chain for basic scheduling, where the local feedback $\epsilon = 0$, with the exception of the mth stage which has a feedback of ϵ. This may be SCkt $m = 3$ in the example of Fig. 7. In this case, the best number of WR iterations is given by *one* iteration up to circuit $m - 1 = 2$. Then, the local iteration between SCkts 2 and 3 should be iterated to convergence. Finally, all the circuits following SCkt 3 again require only *one* iteration. It is evident that a brute-force global analysis using a basic overall schedule would be much more costly than an analysis with the best possible global schedule outlined here.

The general situation for a subcircuit may be very challenging due to the potentially complex interconnections. A SCkt m with its variables x_m corresponding to Eq. (1) is represented with all the potentially connected variables corresponding to other subsystems as

$$\begin{aligned}\sum_{l=1}^{m} C_{ml}(x_1^{(w+1)}, \ldots, x_m^{(w+1)}, x_{m+1}^{(w)}, \ldots, x_M^{(w)})\dot{x}_l^{(w+1)} \\ + \sum_{l=m+1}^{M} C_{ml}(x_1^{(w+1)}, \ldots, x_m^{(w+1)}, x_{m+1}^{(w)}, \ldots, x_M^{(w)})\dot{x}_l^{(w)} \\ - f_m(x_1^{(w+1}, \ldots, x_m^{(w+1)}, x_{m+1}^{(w)}, \ldots, x_M^{(w)}, u) = 0\end{aligned} \tag{36}$$

where the system is of size M.

The updating in this equation is of the GS type. It is clear that the challenge is to partition a large circuit in such a way that similar local convergence factors result for all the variables involved. A relatively uniform basic schedule can then be used without large inefficiencies. This is complicated by the presence of feedback loops. Feedback loops have been investigated by several researchers like Juan and Gear (45) and Johnson and Ruehli (46). The work in Ref. 42 uses a theoretical model, while the work in Ref. 43 is based on numerical experiments. Experimental evidence shows that tight feedback situations, which do not include many subcircuits inside a feedback loop as they exist in flip-flop circuits, lead to a much larger number of WR iterations than loops that involve more SCkts. This is due to the instantaneous and highly nonlinear interactions of the SCkts in tight loops. More details on this issue will be given later in the section entitled "Parallel Waveform-Relaxation-Based Circuit Simulation."

WINDOWING AND OTHER EFFICIENCY IMPROVEMENTS

Windowing

It was shown in the section entitled "Fundamental WR Techniques" that convergence is a function of the analysis time interval. Specifically, the local convergence can be accelerated by subdividing the total analysis time into a series of sequential unequal time windows. All subcircuits need to be solved to convergence within a time window before moving on to the next window. The time window needs to be as large as possible to allow the SCkts to operate with independent time steps such that the multirate factor is maximized. In contrast to this it has been observed in Theorem 4 that the larger the nonlinearities or equivalently the Lipschitz constants K and L, the smaller we must choose the window sizes T. Fortunately, the time step may also be several orders of magnitude smaller during the high-gain nonlinear transition where K and L are large such that the number of time steps per window is not drastically decreased during the highly nonlinear transitions. Hence, we can still expect to obtain a reasonable multirate factor.

Time-windowing algorithms have been suggested by several authors [e.g., (14,19,47)]. Peterson and Mattisson (47) suggest a time-windowing scheme that initially creates windows whenever an input waveform changes state. Then as the analysis proceeds, windows may be truncated based on the convergence rate of the subcircuits and the number of accumulated time points. By limiting the number of time points within a window, memory requirements can easily be managed and controlled.

In general, it is very hard to come up with heuristic windowing algorithms for heterogeneous circuits. The best window size is not only determined by the local convergence rate but also by strong feedback loops such as a flip-flop or a ring oscillator loop. Hence the dynamics of the local situation plays a major role in the choice of the window size as we will illustrate later. We again use the same complementary MOS (CMOS) inverter chain in Fig. 7 for the windowing examples as we did for the partitioning and scheduling. We note from the data in Table 2 that the best results are only weakly dependent on window size. The dependence is much stronger for circuits with a complex fanout structure and for strong feedback situations such as a ring oscillator. In this case, the period of the oscillator determines the best window size, as is shown by Urahama and Kawane (33).

Table 2. Effects of Window Size

Ckt Name	No. Trans.	Analysis time (ns)	Best Window (ns)
Ch8	16	4.5	4.5
Ch16	32	8	2
Ch32	64	16	4

Latency

Efficiency improvements in the WR method have been pursued almost since its beginning. Waveform convergence may be measured by different weighted norms based on $\|\cdot\|_\infty$ or on the $\|\cdot\|_2$ norm, which may lead to a more sensitive criterion. This issue was first reported by Debefve, Hsieh, and Ruehli (48).

Some of the additional convergence testing concepts lead to considerable reduction in compute time. For a large circuit, there usually exist some subcircuits that do not need to be analyzed, because their surrounding subcircuits do not change over a particular window in time T. This situation is stated in the next paragraph in some detail.

For a given subcircuit SCkt, we call all the associated waveforms $x(t)$. They include the external waveforms $x_E(t)$ and the internal waveforms $x_I(t)$, corresponding to nodal voltages or current external or internal to the given subcircuit SCkt, respectively.

Definition 4. A SCkt is said to be latent if

1. The SCkt has been analyzed at least once for the present time window T.
2. All external waveforms $X_E(t)$ associated with the SCkt do not change between iterations (w) and $(w + 1)$ in the present time window T. This change is measured by comparing

$$\|x_E^{(w)}(t) - x_E^{(w-1)}(t)\| \le \epsilon_A + \epsilon_R \max_{t\in[0,T]} \|x_E^{(w-1)}\|_\infty \qquad (37)$$

where ϵ_A is the absolute waveform error and ϵ_R is the relative waveform error.

Then the subcircuit SCkt is declared latent and is not analyzed until either the inputs $x_E(t)$ change or the analysis moves on to a new time window. Essentially, latency is the limiting form of partial waveform convergence considered in the next section. The application of latency can lead to an appreciable improvement in overall solution efficiency. For example, the solution of a 4-bit ALU with 282 FET transistors analyzed on a small IBM RS/6000 workstation required 249 central processing unit (CPU) seconds without the above latency algorithm invoked, as compared with 101 CPU seconds with the latency algorithm used.

Partial Waveform Convergence

This algorithm represents a more elaborate form of latency. It was recognized that many waveforms were rejected toward

the end of the time window T due to the nonuniform convergence of the WR process. This nonuniform convergence was considered earlier. The partial waveform convergence is given by the following algorithm.

Definition 5. A SCkt is said to be partially converged or $\hat{t}$ *partially latent* if

1. The SCkt has been analyzed at least once for the present time window T.
2. All waveforms $x_E(t)$ associated with the SCkt do not change up to the time point $\hat{t}$ for (w) and $(w + 1)$. This change is measured as

$$\|x_E^{(w)}(t) - x_E^{(w-1)}(t)\| \leq \epsilon_A + \epsilon_R \max_{t \in [0,\hat{t}]} \|x_E^{(w-1)}\|_\infty \qquad (38)$$

where ϵ_A is the absolute waveform error and ϵ_R is the relative waveform error.

Then the subcircuit SCkt does not need to be re-solved over the entire interval [0, T] for iteration $(w + 1)$, but only over the shorter interval [$\hat{t}$, T].The application of partial waveform convergence can lead to an appreciable improvement in overall efficiency. For example, the solution of a clock-signal-generation circuit containing 1059 FETs run again on a small IBM RS/6000 workstation required 2861 CPU seconds when partial waveform convergence was not used, versus only 2430 CPU seconds using the partial waveform convergence just mentioned.

Coupled and Preconditioned WR

The WR approach has the potential to be used in many different ways due to its iterative basis. Here, we consider two different important aspects on how a WR circuit solver can interact with other circuit solvers. Several circuit simulators must cooperate together in a multilevel simulation environment. A higher-level simulator may have to be coupled to a WR circuit solver. A multirate waveform interface (49,50) is a very good way to couple tools together by exchanging waveforms during each time window. However, the coupled waveforms may have to be subjected to some processing such that the waveforms fullfill the appropriate smoothness conditions. The WR solver will supply the appropriate master time windows.

Another approach has been proposed by Burrage (15) in which the waveforms are preconditioned with some other waveforms. Very good waveforms may be obtained from a faster more approximate circuit simulator. We did some informal studies of the preconditioning process by distorting the solution waveforms obtained from a WR solver. We discovered two different regimes. Very rapid convergence to the exact waveforms was observed, provided that the distortion was not too large. For the case in which the distortion was large, the starting waveforms seem to have little impact on the convergence behavior. It should be noted that many other situations are relevant. For example, in a hierarchical situation as is shown in Fig. 1 only a few waveforms need to be known at an interface between the functional units to enable the analysis of other functional units using existing waveforms for WR.

PARALLEL WAVEFORM-RELAXATION-BASED CIRCUIT SIMULATION

Parallel implementations of WR have been investigated by many researchers (47,51–57) since the approach is ideally suited for parallelization. Many of the techniques developed for parallel WR are detailed in the book by Banerjee (58). Because each subcircuit is solved independently, subcircuits can be distributed among multiple processors and solved concurrently. During every iteration, each processor must have access to the input waveforms for each subcircuit that it is to solve. Once waveforms are available, a processor can then solve a subcircuit over a time window T. Only after a subcircuit has been solved is there a need to share data among processors. This results in infrequent sharing of relatively large blocks of data among processors. Generally the time to solve each subcircuit is relatively long compared with the time needed to communicate results among processors. This implies that the ratio of time for computation to communication will be high, and good parallel speedups are possible. Moreover, as circuit size increases, the size of each subcircuit often remains relatively constant, while the number of subcircuits generally increases. Therefore as circuit size increases, the opportunities for parallelism also increase.

Architecture Considerations

Parallel-processing machines can be grouped into two classes: single-instruction, multiple-data (SIMD) and multiple-instruction, multiple-data (MIMD). In a SIMD machine, each processor executes the same instructions on different data streams. In a MIMD machine, each processor executes different instructions on different data streams. Parallel WR solves different subcircuits on each processor, and therefore each processor will in general be executing different instructions on different data, which implies that parallel WR is best suited for a MIMD architecture. Additionally, both SIMD and MIMD machines can be implemented using either shared or distributed memory. In a shared-memory machine, each processor is capable of accessing all memory in the machine. It is usually the programmer's responsibility to make sure no two processors attempt to access the same memory locations simultaneously. The Cray C-90™ and SGI IRIS Challenger™ are examples of shared memory MIMD machines. In a distributed memory machine, each processor has its own local memory, which cannot be accessed by other processors. Sharing of data is accomplished through message passing between processors. One form of distributed memory machine is a network of workstations using MPI to share data over a network. The IBM SP2™, Intel Paragon™, and Cray T3D™ are examples of more closely coupled distributed-memory MIMD machines. One advantage of distributed-memory machines is that no single processor needs to have enough memory to hold all of the data for analysis. This becomes increasingly important as circuit sizes increase. On the other hand, shared memory permits faster exchange of data among processors.

As stated above, a MIMD architecture is well suited for WR where parallelism is applied at the subcircuit level with each processor solving its own set of subcircuits. Either shared or distributed memory can be used, each with its own advantages and disadvantages. In a shared-memory environment, it is easier to balance work load among the processors,

because each processor has complete access to all data relative to the analysis. As each processor completes an analysis of a subcircuit, it solves the next subcircuit that is ready to be processed (51). In this way, slower processors will automatically take on less work, while faster processors will do more. One associated disadvantage is that a relatively complicated locking mechanism must be implemented to prohibit different processors from trying to read and write the same data at the same time. Another is that all input data and computed results must fit within the globally shared memory.

Distributed memory eliminates problems relating to simultaneous access of data and the need to have all data fit within one global memory. However, because all data are not easily accessible to all processors, it is harder to balance work load. Most implementations statically assign subcircuits to processors at the beginning of an analysis using a combination of heuristics to attempt to predict and balance work load and communication patterns (56). Dynamic work load balancing (59) requires the transfer of subcircuits and their "state" from one processor to another, which may be several thousands of bytes. If these transfers cannot be done quickly or they must be done often, it may be faster to stay with a suboptimal subcircuit to processor assignment. In addition, performance may be affected by the time required to share data among processors. Fortunately, windowed WR at the subcircuit level requires infrequent sharing of data among processors. Nevertheless, the time to communicate results may be a significant portion of total job time. Consequently, most MIMD implementations attempt to minimize communication by assigning subcircuits that share data to the same processor and to "hide" communication overhead by overlapping communication and computation, that is, by continuing to compute additional results while communication is progressing. The underlying assumption is that parallel WR is applied to very large circuits that partition into many subcircuits, and that there are many more subcircuits than processors. Therefore, each processor will generally have sufficient work to remain active while data are being shared among processors.

Algorithm Selection

It was shown earlier that the GS relaxation algorithm will, in general, converge in fewer iterations than the GJ algorithm, and is usually the favored implementation for sequential processing. However, the faster convergence rate of the GS algorithm is derived from an ordering and scheduling of subcircuits that limits parallelism. Parallelism is limited by the number of subcircuits that can be scheduled at each Seidel level. Circuits that partition into long chains of subcircuits with little fanout will have little parallelism to exploit, whereas circuits like the DRAM error correction circuit shown in Fig. 2 offer a great deal of potential parallelism. In contrast, parallelism using the GJ algorithm is limited only by the number of subcircuits. With the GJ algorithm, during waveform iteration $(w + 1)$ all subcircuits are solved using input waveforms computed during iteration (w). Hence no ordering of subcircuits is necessary. This implies that once all subcircuits have been solved for an iteration, all data are available to schedule all subcircuits for the next iteration. Consequently, the GJ algorithm has the potential for parallelism that is equal to and increases linearly with the number of subcircuits.

Although the parallel GS method will retain a faster convergence rate over the GJ method (fewer iterations), because of the limits on available parallelism, the time to complete those iterations may actually be longer than the time to complete the GJ iterations. If the number of available processors is large, the GJ algorithm will in general be able to use all of them. The GS algorithm, on the other hand, will only be able to use effectively a number of processors equal to the maximum number of subcircuits scheduled at any Seidel level. Therefore, the GS algorithm is not necessarily the best algorithm for parallel processing. However, if the number of processors is smaller than the average number of subcircuits at each Seidel level, then the GS method is probably the better choice. In such cases parallelism will be limited by the number of processors, and the faster convergence rate of the GS algorithm will result in a faster solution. In most applications the number of processors is limited, whereas the number of subcircuits and their relationship to one another is circuit dependent. The best implementation would be to include both algorithms with automatic selection of the GS or GJ algorithms based upon circuit topology and the number of processors available to solve the problem.

Another implementation consideration is memory usage. In order to determine convergence, at any iteration $(w + 1)$, both GS and GJ algorithms require storage to hold computed waveforms for iterations (w) and $(w + 1)$. For each subcircuit, complete waveforms must be retained for all computed waveforms for two iterations. For a single processor, this implies that all waveforms must be stored twice. However, on a multiprocessor system, each processor only needs to store iteration (w) and $(w + 1)$ values for those waveforms that are actually computed on that processor, along with waveforms for either the (w) or $(w + 1)$ iteration of inputs solved on other processors. Input waveforms are needed for iteration (w) when using the GJ algorithm and for iteration $(w + 1)$ when using the GS algorithm. With the GS algorithm, newly computed waveforms can be shared with other processors immediately. However, unless each processor maintains storage for inputs for both iterations $(w + 1)$ and (w), the GJ algorithm must delay sharing newly computed waveforms among processors until all processors have completed each waveform iteration. Otherwise, data for iteration $(w + 1)$ may overwrite data expected to be for iteration (w). Consequently, the parallel GS method can be implemented to use less storage per processor than the GJ method. The alternative is to defer sharing of data until all processors have completed an iteration. This can result in communication bottlenecks and substantially reduce performance, especially for distributed-memory machines.

With the GS algorithm, data must be shared among processors throughout the analysis of a time window in order for the solution to proceed. If input waveforms are not available to solve a subcircuit, a processor may have to wait for data to be computed on another processor. So not only does the GS algorithm limit parallelism, it also may introduce bottlenecks and adversely affect load balance among processors. In an attempt to reduce these effects, Zukowski and Johnson (60) have reported implementation of a "mixed" Seidel–Jacobi or bounded-chaotic algorithm that attempts to solve all subcircuits using the GS algorithm. However, if a processor is idled due to lack of input waveforms for the current iteration, a subcircuit is chosen to be solved using whatever waveforms are available. Some inputs may be from the current, while

others may be from the previous iteration. The algorithm is bounded in that waveforms can be at most one iteration behind the current iteration, like the Jacobi algorithm. The hope is that a solution will be completed faster if processors remain busy, even if all of the input waveforms do not meet strict Seidel ordering. For circuits with large fanouts that permit the effective use of a large number of processors, this implementation retains the faster convergence rate of the GS algorithm. For circuits with less fanout, this technique should take no longer than the GJ algorithm in which all input waveforms are from the previous iteration.

Implementations

Parallel WR may be implemented using either a master–slave or a data-driven paradigm. In a master–slave implementation, one processor serves work to the others and synchronizes each iteration of the analysis. The master–slave setup is well suited for a shared-memory machine, because all data are available to all processors, and therefore the master can quickly assign any work item to any processor without the need to transmit large quantities of data. In addition, the master can maintain data integrity by only permitting one processor to access a specific data item at a time.

In a data-driven implementation, each processor solves its assigned subcircuits as soon as input waveforms are available. Synchronization is required only to determine convergence, update window boundaries, and prepare output files. A data-driven implementation will function equally well on either shared- or distributed-memory machines.

Efficiency Improvements

With either implementation, whenever input data are available to solve a subcircuit, the circuit can be scheduled for analysis. In general, there will be many more subcircuits than processors, and each processor will have more than one subcircuit that can be solved at any time. Under such conditions, a choice must be made as to the order in which the subcircuits are solved. When using the GJ algorithm, the choice is unimportant. However, when using the GS algorithm, this choice may greatly affect overall performance and throughput. The subcircuits for which data are available should be sorted and solved in order based upon the level at which their outputs are needed. For example, consider the situation in which a processor has two subcircuits that can be solved. One has outputs that are needed as inputs to another subcircuit at level 4, while the other subcircuit's outputs are not needed until level 5. The subcircuit whose outputs are needed at level 4 should be solved first. This will permit the outputs to be communicated to other processors while the second subcircuit is being solved. Hopefully the data will arrive before the second processor finishes the subcircuits it is currently solving, and the processor will not have to wait for data.

In the previous section "Ordering and Scheduling," options were discussed for dealing with feedback loops. However, the choice of whether to break feedback loops into SCkts is different when using a multiprocessor system (46). One of the primary goals of parallel WR is to keep all of the processors busy most of the time. Feedback loops that are merged into a single subcircuit maintain strict GS ordering, but they create larger subcircuits. This has a negative impact on load balancing, matrix size, and the multirate speedup. However if feedback loops are cut such that two (or more) similarly sized subcircuits are created, these subcircuits can be distributed among the processors. Since we expect cut feedback loops to result in additional WR iterations, it is advantageous to iterate multiple times during each WR iteration among subcircuits resulting from cut feedback loops. Table 3 gives timing results for 16 circuits ranging in size from under 300 to over 93,000 transistors when all feedback loops are cut versus only cutting long loops where the feedback loop extends over several subcircuits. These results were obtained using the experimental Victor, V256 processor described in Ref. 56, with the larger circuits using all 256 processors.

Table 3. Timing Results

	Time (s)	
Ckt. Name	All Loops Cut	Only Long Loops Cut
Ckt1	28	116
Ckt2	33	162
Ckt3	46	134
Ckt4	90	246
Ckt5	103	242
Ckt6	106	297
Ckt7	113	278
Ckt8	142	175
Ckt9	155	285
Ckt10	159	288
Ckt11	195	355
Ckt12	226	316
Ckt13	229	352
Ckt14	537	815
Ckt15	477	1111
Ckt16	1025	1509

SUMMARY AND CONCLUSIONS

We summarize the state of the waveform-relaxation techniques in this article. WR is a very active area of research as is evident from the publications listed here, which are only a fraction of all the work done in this area. Also, there are many more relevant works on WR that are of interest. To mention just a few topics of interest, there are the faster sensitivity computations by Chen and Feng (61) and the related error measuring technique by Gristede, Zukowski, and Ruehli (62). Other work of importance is hierarchical WR by Saviz and Wing (35). We hope that it is evident from this article that WR is an interesting area of research with potential for further innovations as well as applications.

The WR approach shows a clear speed advantage for very large circuits over conventional circuit solvers. However, even today a fast workstation is required to run circuits that are large enough to show substantial gains. This may be of interest for a large company or to a university, but it is of a lesser interest to the average user of a circuit solver like the many SPICE-like tools that may run on a small machine. We expect that the WR approach will become much more popular with the next generation of high-performance workstations, which include multiple processors at a more moderate price. As is evident from this article, the gains in compute time will be substantial. We expect that the availability of parallel computing for a wider audience will make the WR algorithms of more interest to EDA companies.

BIBLIOGRAPHY

1. L. W. Nagel, *SPICE2, a computer program to simulate semiconductor circuits,* Memo UCB/ERL M520, University of California, Berkeley, May 1975.
2. W. T. Weeks et al., Algorithms for ASTAP—a network analysis program, *IEEE Trans. Circuit Theory,* **CT-20**: 628–634, 1973.
3. R. Saleh, S-J. Jou, and A. R. Newton, *Mixed-mode simulation and analog multi-level simulation,* Norwell, MA: Kluwer, May 1994.
4. C. Visweswariah and R. A. Rohrer, Piecewise approximate circuit simulation, *IEEE Int. Conf. Comput.-Aided Design ICCAD,* November 1989.
5. A. Devgan and R. A. Rohrer, Adaptively controlled explicit simulation, *IEEE Trans. Comput.-Aided Des. IC's Syst.,* **CAD-13**: 746–762, 1994.
6. A. E. Ruehli, A. L. Sangiovanni-Vincentelli, and G. Rabbat, Time analysis of larger scale circuits containing one-way macromodels, *IEEE Trans. Circuits Syst.,* **CAS-29**: 185–191, 1982.
7. E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli, The waveform relaxation method for the time-domain analysis of large scale integrated circuits, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **CAD-1**: 131–145, 1982.
8. J. White and A. Sangiovanni-Vincentelli, Relax2: A Modified Waveform Relaxation Approach to the Simulation of MOS Digital Logic, *IEEE Proc. Int. Symp. Circuits Syst.,* Newport Beach, CA, 1983.
9. J. F. Beetem et al., A large scale mosfet circuit analyzer based on waveform relaxation, *Conf. Comput. Des.,* 1984, pp. 507–514.
10. U. Miekkala and O. Nevanlinna, Convergence of waveform relaxation method, *IEEE Int. Conf. Circuits Syst., ISCAS-88,* 1988, pp. 1643–1646.
11. A. Lumsdaine and J. White, Accelerating waveform relaxation methods with applications to parallel semiconductor device simulation, *Numer. Funct. Anal. Optimiz.,* **16** (3+4): 395–414, 1995.
12. S. Vandewalle and R. Piessens, Numerical experiments with nonlinear multigrid waveform relaxation on parallel processor, *Appl. Numer. Math.,* **8**: 149–161, 1991.
13. B. Leimkuhler, Estimating waveform relaxation convergence to parallel semiconductor device simulation, *SIAM J. Sci. Comput.,* **14**: 872–889, 1993.
14. P. Debefve, F. Odeh, and A. Ruehli, in A. Ruehli (ed)., *Circuit Analysis, Simulation and Design,* New York: Elsevier, North-Holland, 1987.
15. K. Burrage, *Parallel and Sequential Methods for Ordinary Differential Equations,* Oxford, UK: Oxford Science, 1995.
16. C.-W. Ho, A. Ruehli, and P. Brennan, The modified nodal approach to network analysis, *IEEE Trans. Circuits Syst.,* **CAS-22**: 504–509, 1975.
17. K. R. Schneider, A remark on the waveform relaxation method, *Int. J. Circuit Theory Appl.,* **19**: 101–104, 1991.
18. J. White et al., Waveform relaxation: Theory and practice, *Trans. Soc. Comput. Simulation,* **2** (2): 95–133, June 1985.
19. J. White and A. Sangiovanni-Vincetelli, *Relaxation Techniques for the Simulation of VLSI Circuits,* Norwell, MA: Kluwer, 1987.
20. C. H. Carlin and A. Vachoux, How to Use Underrelaxation to Improve Waveform Relaxation Convergence, *Proc. Int. Conf. Comput.-Aided Des. ICCAD-87,* November 1987.
21. U. Miekkala and O. Nevanlinna, Sets of convergence and stability regions, *BIT,* **27** (4): 554–584, 1987.
22. M. Reichelt, J. White, and J. Allen, Waveform Relaxation for Transient Two-Dimensional Simulation of MOS Devices, *Proc. Int. Conf. Comput.-Aided Des. ISCAS-89,* November 1989, pp. 412–415.
23. M. P. Desai and I. N. Hajj, On the convergence of block relaxation methods for circuit simulation. *IEEE Trans. Circuits Syst.,* **36**: 948–958, 1989.
24. G. D. Gristede, A. E. Ruehli, and C. A. Zukowski, Convergence properties of waveform relaxation based circuit simulation methods, to appear in *IEEE Trans. Circuits Syst.* **45** (7): July, 1998.
25. U. Miekkala, O. Nevanlinna, and A. E. Ruehli, Convergence and Circuit Partitioning Aspects for Waveform Relaxation, in D. W. Walker and Q. F. Strout (eds.), *Proc. 5th Distrib. Memory Comput. Conf.,* Los Alamitos, CA: IEEE Comput. Soc. Press, 1990, pp. 605–611.
26. B. Leimkuhler, U. Miekkala, and O. Nevanlinna, Waveform relaxation for linear RC circuits, *Impact Comp. Sci. Eng.,* **3** (2): 123–145, 1991.
27. A. E. Ruehli and C. A. Zukowski, Convergence of Waveform Relaxation for RC Circuits, in X. Odeh et al., *Semiconductors* Part I-II), IMA Volumes in Mathematics and Its Applications, New York: Springer, 1992.
28. B. Leimkuhler and A. Ruehli, Rapid convergence of waveform relaxation, *Appl. Numer. Math.,* **11** (3): 211–224, 1993.
29. C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations,* Englewood Cliffs, NJ: Prentice-Hall, 1971.
30. K. J in 't Hout, On the convergence of waveform relaxation methods for stiff nonlinear ordinary differential equations, *Appl. Numer. Math.,* **18**: 175–190, 1995.
31. K. Taubert and W. Wiedl, Waveform relaxation und ein Störungssatz für akkretive operatoren in der maximumnorm, *ZAMM Z. angew. Math. Mech.,* **73** (7/8): 925–927, 1993.
32. W. M. B. van Bokhoven, An Activity Controlled Modified Waveform Relaxation Method, *IEEE Proc. Int. Conf. Circuits Syst., ISCAS,* 1983, pp. 765–768.
33. K. Urahama and Y. Kawane, Circuit simulation using event-driven waveform relaxation-Newton method. *Trans. IEICE,* **71** (1): 1–3, 1989.
34. D. J. Erdman and D. J. Rose, A Newton Waveform Relaxation Algorithm for Circuit Simulation, *Proc. Int. Conf. on Comput.-Aided Des.,* November 1989, pp. 404–407.
35. P. Saviz and O. Wing, Pyramid—a hierarchical waveform relaxation-based circuit simulation program, *IEEE Int. Conf. Comput.-Aided Des.,* 1988, pp. 442–445.
36. C. H. Carlin and A. Vachoux, On Partitioning for Waveform Relaxation Time-Domain Analysis of VLSI Circuits, *IEEE Proc. Int. Symp. Circuits Syst. ISCAS,* 1984, pp. 701–705.
37. J. White and A. L. Sangiovanni-Vincentelli, Partitioning Algorithms and Parallel Implementations of Waveform Relaxation Algorithms for Circuit Simulation, *IEEE Proc. Int. Symp. Circuits Syst. ISCAS-85,* 1985, pp. 1069–1072.
38. T. J. Cockerill, et al., Toggle: A Circuit Analyzer for MOSFET VLSI, *Proc. Int. Conf. VLSI and Comput. COMP-EURO'87, Hamburg, Germany,* May 1987.
39. W. John, W. Rissiek, and K. L. Paap, Circuit Partitioning for Waveform Relaxation, *Proc. Eur. Des. Automat. Conf. EDAC,* The Netherlands, February 1991, pp. 149–151.
40. D. Dumlugol, J. Cockx, and H. DeMan, Segmented waveform analysis for computation of accurate starting waveforms in circuit simulation, *Tech. Report Katholieke Universitiet Leuven,* 1984.
41. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms,* Reading, MA: Addison-Wesley, 1983.
42. M. E. Mokari-Bolhassan, D. Smart, and T. N. Trick, A New Robust Relaxation Technique for VLSI Circuit Simulation, *IEEE Proc. Int. Symp. Circuits Syst., ISCAS-85,* November 1985.

43. R. Jeltsch and B. Pohl, *Waveform relaxation with overlapping systems,* Report No. 91-02, Seminar für Angewandte Mathematik, ETH Zürich, 1991.
44. F. Odeh, A. Ruehli, and C. H. Carlin, Robustness Aspects of an Adaptive Waveform Relaxation Scheme, *IEEE Proc. Int. Conf. Comput. Des., ICCD,* Rye, NY, October 1983, pp. 396–399.
45. F. L. Juan and C. W. Gear, *Accuracy increase in waveform Gauss Seidel,* Comp. Sci. Dept. Report 1518, University of Illinois, Urbana-Champain, 1989.
46. T. A. Johnson and A. E. Ruehli, Parallel Waveform Relaxation of Circuits with Global Feedback Loops, *Proc. 1992 Des. Automat. Conf.,* Anaheim, CA, June 1992, pp. 12–15.
47. L. Peterson and S. Mattison, The design and implementation of a concurrent circuit simulation program for multicomputers, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **12**: 1004–1014, 1993.
48. P. Debefve, H. Y. Hsieh, and A. E. Ruehli, Wave Convergence Algorithms for the Waveform Relaxation Method, *IEEE Proc. Int. Conf. Comput. Aided Des., ICCAD,* November 1984.
49. S. Skelboe, Stability properties of backward differentiation multirate formulas, *Appl. Numer. Math.,* **5**: 151–160, 1989.
50. M. Günther and P. Rentrop, Partitioning and multirate strategies in latent multirate electric circuits, *Int. Ser. Numer. Math.,* **117**: 33–60, 1994.
51. D. Smart and T. Trick, Increasing Parallelism in Multiprocessor Waveform Relaxation, *Proc. Int. Conf. Comput.-Aided Des. ICCAD-87,* November 1987, pp. 360–363.
52. A. Sangiovanni-Vincentelli, *Parallel Processing for Simulation of VLSI Circuits,* Amsterdam: Elsevier,1988.
53. S. Raman and L. M. Patnaik, HIRECS: Hypercube implementation of relaxation-based circuit simulation, *Int. J. High Speed Comput.,* **1** (3): 399–432, 1989.
54. R. Saleh et al., Parallel circuit simulation on supercomputers, *Proc. IEEE,* **77**: 1915–1931, 1989.
55. P. Odent, L. Claesen, and H. DeMan, Acceleration of relaxation-based circuit simulation using a multiprocessor system, *IEEE Trans. Comput.-Aided Des.,* **9**: 1063–1072, 1990.
56. T. A. Johnson, Waveform-relaxation-based circuit simulation on the Victor V256 parallel processor, *IBM J. Res. Devel.,* **35** (5): 707–720, 1991.
57. A. D. Jalnapurkar, C. D. McCrosky, and H. C. Wood, Simulation of MOS VLSI circuits using parallel processors, *1993 Wescanex,* 1993, pp. 171–176.
58. P. Banerjee, *Parallel Algorithms for VLSI Computer-Aided Design,* Englewood Cliffs, NJ: Prentice-Hall, 1994.
59. L. Peterson and S. Mattisson, Dynamic Partitioning for Concurrent Waveform Relaxation Based Circuit Simulation, *1993 IEEE Int. Symp. Circuits Syst.,* May 1993, pp. 1639–1642.
60. D. Zukowski and T. A. Johnson, Efficient Parallel Circuit Simulation Using Bounded-Chaotic Relaxation, *1992 IEEE Int. Symp. Circuits Syst.,* May 1992, pp. 911–914.
61. C.-J. Chen and W.-S. Feng, Transient Sensitivity Computations of MOSFET Circuits Using Iterated Timing Analysis and Selective-Tracing Waveform Relaxation, *Proc. 1994 Des. Automat. Conf.,* San Diego, CA, June 1994, pp. 581–585.
62. G. D. Gristede, C. A. Zukowski, and A. E. Ruehli, Measuring error propagation in waveform relaxation algorithms, *IBM Res. Report,* June 1997.

ALBERT E. RUEHLI
IBM Research Division

THOMAS A. JOHNSON
IBM Microelectronics Division