# THE WAVEFORM RELAXATION METHOD FOR TIME DOMAIN ANALYSIS OF LARGE SCALE INTEGRATED CIRCUITS: THEORY AND APPLICATIONS

by

Ekachai Lelarasmee

Memorandum No. UCB/ERL M82/40

19 May 1982

## Abstract

The Waveform Relaxation (WR) method is a new decomposition method for solving a system of mixed implicit algebraic-differential equations over a given time interval. This method essentially uses an iterative relaxation scheme such as the Gauss-Seidel relaxation or the Gauss-Jacobi relaxation in which the elements of the relaxation are waveforms of unknown variables. The decomposed system obtained through the relaxation scheme is also a system of mixed implicit algebraic-differential equations but is much easier to solve than the original system.

The application of this method in the area of time domain simulation of integrated circuits is emphasized. Although the WR method has a theoretical basis, it can be given a simple physical interpretation when applied to the analysis of integrated circuits. In particular, the convergence conditions of the method can be given either in terms of the numerical properties of the circuit equations or in terms of the properties of the circuit components. This method is shown to be a viable alternative to the conventional techniques for simulating large scale integrated circuits since sufficient conditions for its convergence are quite mild and are always satisfied by a large class of practical circuits.

The performance of the WR method when applied to a particular class of circuits, i.e., MOS digital integrated circuits, are studied and evaluated via a prototype simulator called RELAX. The repetivity and directionality of digital subcircuits as well as the digital nature of the signals are exploited in the simulator to increase the speed of computation and to utilize the storage efficiently. Preliminary comparisons between RELAX and the standard circuit simulator SPICE have shown that RELAX is fast and reliable for simulating MOS digital integrated circuits.

# Acknowledgements

I am deeply grateful to my research advisor, Professor Alberto L. Sangiovanni-Vincentelli, who introduced me into the exciting area of computer-aid design and optimization and has consistently guided me with enthusiasm throughout the course of my graduate study.

I appreciate the scholarship from the Anandamahidol Foundation (Bangkok, Thailand) which has given me the financial support in persuing my graduate study here at U.C. Berkeley. I also appreciate the research grants from Harris Semiconductor Corporation, IBM Corporation and Joint Services Electronic Program contract F49620-79-C-0178.

I wish to express my sincere thanks to Professor A.R. Newton, Dr. Albert E. Ruehli from IBM Research Center and J.P. Spoto from Harris Semiconductor for many useful and stimulating discussions necessary for the progress of this work and to the DELIGHT creator William T. Nye for numerous assistances in programming. I also wish to express the pleasure of my association with many friends: Dr. M.J. Chen, Dr. V.H.L. Cheng, G. DeMicheli, C.L. Gustafson, R.J. Kaye, M. Lowy, W.T. Nye, D.C. Riley, Dr. V. Visvanathan and the Projectile people of Ridge Project.

Finally, I wish to express my special gratitude for the inspiration and caring provided by my dearest parents: Yothin and Yuwadee.

# Table of contents

# Chapter 1

# Introduction

Simulation programs have proven to be effective software tools in evaluating or verifying the performance of integrated circuits during the design phase. Circuit simulators such as SPICE [1] and ASTAP [2] have been widely used by circuit designers to provide accurate electrical analysis of the circuits being designed. Although these simulators are designed to perform many types of analysis such as "dc" analysis, small signal (or "ac") analysis and time domain (or "transient") analysis, the majority of the use of these simulators in present day circuit design is in the latter area of time domain analysis, the most complicated and expensive type of analysis.

In this dissertation, we shall focus on *time domain* or *transient* circuit simulation. This type of simulation involves the solution of a system of differential equations describing the circuit. The most common approach to solving the circuit equations in time domain analysis consists essentially of the use of three basic numerical methods: an implicit integration method, the Newton-Raphson method and the sparse Gaussian Elimination method. We refer to this approach as the *standard* simulation approach. Circuit simulators that use this standard approach (such as SPICE and ASTAP) are called *standard circuit simulators*. The bulk of the storage and computation of the standard simulation approach lies in the process of *formulating* and *solving* a system of linear algebraic equations simultaneously. It turns out [3] that both the storage and computer time required by standard circuit simulators grow rapidly as the size of the circuit, measured in terms of circuit components, increases. Hence, the cost-effective use of standard circuit simulators for performing transient simulation has been generally limited to circuits having a few hundred devices (e.g.

transistors) or less.

As we move into the era of VLSI (Very Large Scale Integrated) circuits, the demand for simulating larger and larger circuits is continuously growing. It is clear that to simply extend simulation techniques used by standard simulators to circuits containing over 10,000 devices is not practical. Hence new algorithms and simulators must be developed. A survey of these algorithms is given in [4]. These new algorithms include, for example, Block LU factorization, the Tearing Algorithm for solving linear algebraic equations, the Multilevel Newton-Raphson algorithm and timing simulation algorithms. A common theme underlying all these algorithms is the use of *large scale system decomposition.*

The purpose of this dissertation is to introduce another decomposition method for time domain simulation. This method is called the *Waveform Relaxation* (WR) method. The idea behind the development of this method originated from a study of the work of Newton [5] who formulated the timing simulation algorithm in the form of a relaxation technique for solving the nonlinear algebraic equations associated with the discretization of the circuit differential equations. In the WR method, relaxation decomposition is applied at the level of differential equations whereas, in other previously proposed decomposition methods, decomposition is applied at the level of (linear or nonlinear) algebraic equations. Both theoretical and computational aspects of the WR method will be discussed in detail. In particular, the development of an experimental program for simulating MOS digital integrated circuits based on the WR method is described. The program is named RELAX. Preliminary tests of the program and its performance comparison with SPICE indicate that the WR method is highly suitable for analysing this type of large scale integrated circuits.

The organization of this dissertation is as follows. In Chapter 2, we give a brief review of the standard simulation approach and a comprehensive discus-

sion and classification of various decomposition techniques. The rest of this thesis can be subdivided into two parts. The first part, consisting of chapters 3, 4 and 5, describes the WR method and its numerical properties in a purely mathematical context. The second part, consisting of chapters 6 to 9, deals with the WR method in a circuit simulation context. A brief description of these two parts is given below.

In Chapter 3, the mathematical description of the WR method together with the concepts of a *decomposed system* and the *assignment-partitioning process* are given in Chapter 3. In Chapter 4, the effect of the assignment-partitioning process of the WR method on the dynamical behaviour of the decomposed system is described and the concept of *consistency* of the assignment-partitioning process is presented. An algorithm based on graph theory to produce a consistent assignment-partitioning process is also described. In Chapter 5, convergence properties of the WR method are fully discussed by using contraction mappings in functional spaces. Sufficient conditions for convergence of the WR method are given and convergence of the WR method using an adaptive error control mechanism is also discussed.

We begin the second part of this thesis by specializing the WR method to the analysis of VLSI MOS circuits in Chapter 6. Two WR algorithms are described and are shown to converge under very mild and realistic assumptions. In Chapter 7, the details of a few important techniques in implementing the WR method in RELAX are given. The organization of the program is described in Chapter 8 and its experimental results are given in Chapter 9.

Finally, the proofs of all theorems and lemmas are given in Appendix A and in Appendix B we explore the use of iterative techniques to improve the numerical properties of timing simulation algorithms.

# Chapter 2

# Overview of Simulation Techniques

Time domain simulation of a continuous dynamical system, such as an integrated circuit, traditionally uses three basic (or conventional) numerical methods.

a) An implicit integration method which approximates the time derivative operator with a divided difference operator.

b) The Newton-Raphson (NR) method for solving a system of nonlinear algebraic equations.

c) The Gaussian Elimination (GE) method for finding the solution of a system of linear algebraic equations.

The integration method transforms ordinary differential equations into a discrete time sequence of algebraic equations. If the differential equations are nonlinear, the discretized algebraic equations are also nonlinear and can be solved by the NR method. The NR method in turn transforms nonlinear algebraic equations into a sequence of linear algebraic equations which is solved by the GE method. This hierachical organization of numerical methods is shown in Fig. 2.1.

When certain structural and/or numerical properties of a given system of equations are met, the system can be solved efficiently by using the so called *decomposition* techniques. By decomposition, we mean any technique that allows several subsets of the given equations to be solved individually by using conventional numerical methods. In our opinion, decomposition is indispensible in simulating efficiently large scale dynamical systems such as large scale integrated circuits. Various decomposition techniques have been proposed in the circuit simulation literature. A survey of these techniques is given in [4].

In this chapter, we will briefly review the direct applications of the conventional numerical methods in what we call *standard* simulators, such as SPICE [1] and ASTAP [2]. Then we will describe the basic concepts and properties of two fundamental approaches to achieving system decomposition, which have led to the development of several simulators such as SPLICE [5], MOTIS [6], MACRO [7], SLATE [8], DIANA [9], SAMSON [10] and CLASSIE [11].

## 2.1. Standard Simulators.

We define a standard simulator to be a simulator that directly applies the conventional numerical methods (i.e., an implicit integration method, the NR method and the GE method) to the solution of the system of equations describing the behaviour of the circuit to be simulated. Typically, the circuit equations can be written in the following form

$$f(\dot{x}(t), x(t), u(t)) = 0 \; ; \qquad x(0) = x_0 \tag{2.1}$$

where $x(t) \in \mathbf{R}^n$ is the vector of the unknown circuit variables, $u(t) \in \mathbf{R}^r$ is the vector of the independent (or input) variables, $x_0 \in \mathbf{R}^n$ is the given initial value of $x$ and $f : \mathbf{R}^n \times \mathbf{R}^n \times \mathbf{R}^r \rightarrow \mathbf{R}^n$ is a continuous function. Let $\{t_i \; ; \; i = 0,1,\ldots,N\}$ denote a sequence of increasing *timepoints* selected by the simulator with $t_0 = 0$ and $t_N = T$ where $T$ is the given simulation time interval.

By applying an implicit integration method, the system of equations (2.1) is transformed into a discrete time sequence of algebraic equations by replacing $\dot{x}(t_i)$ with an approximating formula

$$\dot{x}(t_i) \approx \mathbf{A}(x(t_i)) \qquad i = 1,2,\ldots N \tag{2.2}$$

Hence, the resulting algebraic equations at time $t_i$ can be written as

$$f(\mathbf{A}(x_i), x_i, u_i) \triangleq g(x_i) = 0 \tag{2.3}$$

where $x_i$ denotes the computed value of $x(t_i)$ and $u_i \triangleq u(t_i)$.

The fact that the approximating formula can be chosen in a variety of ways gives rise to a number of integration methods with different numerical properties, i.e., order of consistency convergence and stability [12]. The most commonly used approximating formulae in the circuit simulation are the *Backward Differentiation* (BD) formulae of order 1 to 6 [12] and the Trapezoidal formula [12]. For example, the first order BD formula, also known as the Backward Euler (BE) formula, is given by

$$\mathbf{A}(x(t_i)) = \frac{x(t_i) - x(t_{i-1})}{t_i - t_{i-1}} \tag{2.4a}$$

and the Trapezoidal formula is given by

$$\mathbf{A}(x(t_i)) = 2\,\frac{x(t_i) - x(t_{i-1})}{t_i - t_{i-1}} - \dot{x}(t_{i-1}) \tag{2.4b}$$

To advance the timepoint, the timestep $h_i \triangleq t_i - t_{i-1}$ is normally selected to ensure that the *local truncation error* [12] associated with the approximating formula is within the prescribed tolerance. The computation of the local truncation error requires that (2.3) be solved accurately.

The solution of (2.3) is obtained in a standard simulator by directly applying the NR method. To start the NR iteration, an initial guess $x_i^0$, called a *predictor*, of the solution is obtained through a prediction step that uses the information of the past trajectories of $x$. For example, a simple linear extrapolation of the past trajectories gives the following predictor (also known as the Forward Euler predictor).

$$x_i^0 = x_{i-1} + \frac{h_i}{h_{i-1}}(x_{i-1} - x_{i-2}) \tag{2.5}$$

The iteration equation of the NR method is given by

$$\frac{\partial g}{\partial x}(x_i^{k-1})[x_i^k - x_i^{k-1}] = -g(x_i^{k-1}) \qquad k = 1,2,... \tag{2.6}$$

where $\frac{\partial g}{\partial x}(x_i^{k-1})$ denotes the Jacobian matrix of $g$ evaluated at $x_i^{k-1}$ and $k$ denotes the NR iteration count. The NR iteration is carried out until the convergence is achieved.

The solution of (2.6) is obtained in a standard simulator by directly applying the GE method. In circuit simulation environments, the coefficient matrix of (2.6) is usually very sparse, i.e., the matrix $\frac{\partial g}{\partial x}$ has very few nonzero elements per row. Hence the GE method is usually implemented in standard simulators by using sparse matrix techniques [1,28]. It is important to exploit the sparsity of (2.6) since the computational complexity of the GE method applied to an $n \times n$ full matrix is proportional to $n^3$ whereas the computational complexity of the GE method using sparse matrix techniques is on the average [1] proportional to $n^\alpha$ ; $\alpha \in [1.2,1.5]$.

Standard circuit simulators such as SPICE [1] and ASTAP [2] have proven to be reliable and effective when the size of the circuit, measured by the number of circuit components, is small. As the size of the circuit increases, the primary storage and computer time used by these simulators increase rapidly [3] despite the use of sparse matrix techniques. It has been estimated [13] that the simulation of a circuit containing 10,000 Metal-Oxide-Semiconductor (MOS) transistors from $t=0$ to $t=1000ns$, using SPICE on an IBM 370/168 computer, would take at least 30 hours of computer time. Hence, the cost effective use of standard circuit simulators has been limited to circuits which are considered small in today VLSI technology.

## 2.2. Decomposition.

Decomposition refers to any technique that subdivides the problem of solving a system of equations into several subproblems. Each subproblem corresponds to solving a subset of equations, called a *subsystem*, for a subset of the system variables. Decomposition can be applied at any level of equations, i.e., differential equations, nonlinear algebraic equations and linear algebraic equations. In effect, the system of equations, no matter at what level it is, is viewed by a decomposition technique as a composition of several subsystems (of the same level of equations) with interactions between them. When the system is decomposed into subsystems, the solution of each subsystem is in general carried out by using the conventional numerical techniques that we have described earlier in the previous section.

There are two different approaches to achieving system decomposition, namely the *tearing* approach and the *relaxation* approach. These two approaches are characterized by different ways of updating the interactions between subsystems and by different numerical properties. Tearing is the approach that aims at exploiting the block structure of the system to achieve decomposition while maintaining the numerical properties of the numerical method that is used to solve the system. Hence, the computational complexity of this approach depends critically on the structure of the system. Clearly, this approach does not provide any gain over conventional numerical techniques when the system structure is not sparse or when the block structure of the system cannot be exploited. On the other hand, relaxation is the approach that decomposes the system into subsystems so as to reduce the complexity of the solution of the decomposed system regardless of whether the system structure is sparse or not, i.e., the decomposed system is always easier to be solved than the original one. However, the numerical properties of this approach are com-

pletely governed by the relaxation scheme, not by the numerical method used to solve the subsystems. These two approaches will be described in more details in the next sections.

In describing the structure of a system of equations, it is customary to introduce the notion of a *dependency* matrix defined as follows.

**Definition 2.1** The dependency matrix $D \in \{0,1\}^{n \times n}$ associated with a system of $n$ equations in $n$ unknown variables is a matrix whose $i,j$-th element $D_{ij}$ is defined by

$$D_{ij} = \begin{cases} 1 & \text{if the } i\text{-th equation involves the } j\text{-th variable} \\ 0 & \text{otherwise} \end{cases}$$

■

The main advantages of using decomposition techniques are:

a) The structural regularity and repetivity of the subsystems, such as those encountered in large scale integrated circuits, can be exploited.

b) Additional savings in computing time can be achieved by incorporating bypassing schemes [3,5,7,8,10] that exploits *latency* or *dormancy* of a subsystem. These schemes allow a simulator to avoid solving a subsystem when its solution can be cheaply predicted within a reasonable accuracy.

c) Decomposition techniques are suitable for computers with parallel or pipeline architectures since more than one subsystem can be solved concurrently.

## 2.2.1. Tearing Decomposition.

Tearing is an approach that exploits the sparsity structure of the dependency matrix of the system to be solved. The particular structures[1] that are suitable for the tearing decomposition are

a) the Bordered Block Diagonal (BBD) structure as shown in Fig. 2.2a.

b) the Bordered Block Lower Triangular (BBLT) structure as shown in Fig. 2.2b.

From these two structures, we see that if the variables associated with the borders of the matrices are known, then the values of the remaining variables can be easily obtained by solving separately the subsystems associated with the diagonal blocks. For this reason these variables are called the *tearing variables*. However, in the tearing approach, the values of the tearing variables are not computed (or updated if the algorithm associated with the tearing decomposition is iterative) from the subset of equations identified by the last diagonal block of the dependency matrix.[2] Instead, they are computed from another subset of equations, called a *reduced subsystem*, which has to be constructed by an algorithm as we shall see later. The number of equations in the reduced subsystem is equal to the number of the tearing variables.

Tearing decomposition of linear algebraic equations can be implemented in two different ways, namely the *Block LU Factorization* [15] and the *Tearing Algorithm* [15].[3] To illustrate the basic ideas behind these algorithms, consider the system of equations shown in Fig. 2.3, i.e.,

$$Ax = b$$

---

[1] In showing a matrix structure, all nonzero elements are confined to the shaded areas only. The shaded areas, however, may themselves contain some zero elements.

[2] We shall see that, in the relaxation approach, these variables will be computed from this subset of equations.

[3] Note that George [14] has interpreted the Tearing Algorithm as a particular form of the Block LU Factorization. We prefer to keep these two algorithms separated to give a better intuitive feeling of the main ideas of these algorithms.

where $A \in \mathbf{R}^{n \times n}$ is in BBD form as shown in Fig. 2.3, $x = \begin{bmatrix} v \\ w \end{bmatrix} \in \mathbf{R}^n$ is the vector of the unknown variables and $w$ is the vector of the tearing variables.

In the Block LU Factorization, the variable $v$ is first eliminated from the system of equations to obtain the following reduced subsystem from which the value of the tearing variable $w$ is obtained.

$$(E - DB^{-1}C)w = b_w - DB^{-1}b_v$$

where the meanings of all matrices and vectors are given in Fig. 2.3. The computed value of $w$ is then used to compute the value of $v$ blockwise.

In the Tearing Algorithm, the solution is obtained by applying the Sherman-Morrison-Woodbury formula [16]

$$x = \hat{A}^{-1}b - \hat{A}^{-1}G[I + H\hat{A}^{-1}G]^{-1}H\hat{A}^{-1}b$$

where the meanings of all matrices are also given in Fig. 2.3 ($F$ in the figure is a nonsingular matrix). Here the solution of the reduced subsystem involves the process of formulating and inverting the reduced system matrix $I + H\hat{A}^{-1}G$ whose size is equal to the dimension of $w$. The full details of the implementations of both tearing decomposition algorithms are given in [15].

The use of tearing decomposition in solving a system of nonlinear algebraic equations gives rise to an iterative method called the *Multilevel Newton-Raphson* (MLNR) method [7]. We briefly describe this method with the help of an example. Consider the problem of computing the "dc" solution of the circuit in Fig. 2.4a. Assume that the circuit equations can be written as

$$f_1(x_1, v_1) = 0 \qquad (2.7a)$$

$$f_2(x_2, v_1) = 0 \qquad (2.7b)$$

$$i_1(x_1, v_1) + i_2(x_2, v_1) = 0 \qquad (2.7c)$$

where $x_1 \in \mathbf{R}^n$ is the vector of all internal variables of the first subcircuit,

$x_2 \in R^{n+1}$ is the vector of all internal variables of the second subcircuit including $v_2$, $f_1 : R^n \times R \to R^n$ describes the equations associated with the first subcircuit, $f_2 : R^{n+1} \times R \to R^{n+1}$ describes the equations associated with the second subcircuit, $i_1 : R^n \times R \to R$ and $i_2 : R^{n+1} \times R \to R$. From this set of equations we see that the output voltage $v_1$ of the first subcircuit is the tearing variable. The decomposed (or torn) circuit is shown in Fig. 2.4b. In the MLNR method, the reduced subsystem is constructed by treating $x_1$ and $x_2$ as functions of $v_1$, i.e., from (2.7a) and (2.7b)

$$x_1 = g_1(v_1) \qquad \text{and} \qquad x_2 = g_2(v_1)$$

and substituting them into (2.7c). Thus the reduced subsystem has the following form

$$i_1(g_1(v_1), v_1) + i_2(g_2(v_1), v_1) \triangleq g(v_1) = 0 \tag{2.8}$$

The reduced subsystem (2.8) is then solved for the tearing variable $v_1$ by using the NR method which yields the following iterative equation

$$\frac{\partial g}{\partial v_1}(v_1^{k-1})[v_1^k - v_1^{k-1}] = -g(v_1^{k-1}) \qquad k = 1,2,... \tag{2.9}$$

where the evaluations of $g(v_1^{k-1})$ and $\frac{\partial g}{\partial v_1}(v_1^{k-1})$ are performed by applying another level of the NR method to (2.7a) and (2.7b). The full details of the implementation of this method is given in [7]. In circuit terms, the construction of the reduced subsystem can be interpreted as replacing each subcircuit by an equivalent (Thevenin or Norton) circuit which is referred to in [7] as the *exact macromodel*. The reduced subsystem is thus equivalent to the interconnection of these exact macromodels. For example, the reduced circuit associated with the circuit equation (2.7) is shown in Fig. 2.5.

Examples of circuit simulators that use tearing decomposition are:

a) CLASSIE [11], SLATE [8] and SAMSON[4] [10]. These simulators implement the Block LU Factorization in the solution of the linear algebraic equations.

b) MACRO [7]. This simulator implements the MLNR method for solving the nonlinear algebraic equations.

Note that whereas the original system of equations may be sparse, i.e., its dependency matrix has a small percentage of nonzero elements, the reduced subsystem may not. Hence the computational advantage of this approach over the standard approach depends crucially on how small each decomposed subsystem and the reduced subsystem are. However, the numerical properties of the tearing approach are the same as those of the standard numerical methods applied to the system without using decomposition. In fact, for linear algebraic systems, both the Block LU Factorization and the Tearing Algorithm give the solution in a finite number of steps since the solution of the reduced subsystem gives the exact values of the tearing variables. For nonlinear algebraic systems, the MLNR method still has the same local quadratic rate of convergence as that of the conventional NR method.

## 2.2.2. Relaxation Decomposition.

Decomposition of a system into subsystems by relaxation is not restricted or fixed by the block structure of the dependency matrix of the system. There is no special procedure for constructing the reduced subsystem in order to solve for the tearing variables as in the tearing approach. The system of equations is simply partitioned into subsystems of equations. Within each subsystem, the variables to be solved for are called *internal* variables and the other variables involving in the subsystem are called *external* variables. If every

[4] SAMSON also implements a blockwise relaxation technique for solving the nonlinear algebraic equations. It is an example of using decomposition techniques at different levels of equations in the same simulation.

subsystem has only one internal variable (or equivalently one equation), the decomposition is said to be done *pointwise*. Otherwise, it is said to be done *blockwise*.

To solve a subsystem for its internal variables, the values of its external variables (which are internal variables of other subsystems) are simply guessed or updated (through an iterative procedure), i.e., the subsystem is *decoupled* or decomposed. This approach usually requires an iterative procedure for repeatedly solving the decomposed subsystems so that the values of the external variables of each subsystem can be updated by using the information from the current or previous iterations. Two well known types of relaxation are the *Gauss-Seidel* (GS) [17] relaxation and the *Gauss-Jacobi* (GJ) [17] relaxation. Fig. 2.6 gives examples of the use of relaxation decomposition at different levels of equations where $k$ denotes the iteration count. Fig. 2.7 shows how to associate relaxation with the conventional numerical methods at different levels of equations in the hierachical organization of a time domain simulation.

Unfortunately this approach does not guarantee that the sequence of iterated solutions will converge to the exact solution of the given system unless a certain numerical condition on the partitioned system is satisfied. This condition is called the *convergence condition* of the relaxation iteration.

As an example, consider the following linear algebraic equations

$$Ax = b \tag{2.10}$$

where $x \in \mathbf{R}^n$ is the vector of the unknown variables, $A \in \mathbf{R}^{n \times n}$ and $b \in \mathbf{R}^n$. Let

$$A = L + D + U \tag{2.11}$$

where $L \in \mathbf{R}^{n \times n}$ is a strictly[5] lower triangular matrix, $D \in \mathbf{R}^{n \times n}$ is a diagonal matrix and $U \in \mathbf{R}^{n \times n}$ is a strictly upper triangular matrix.

[5] A strictly (upper or lower) triangular matrix is a triangular matrix with zero diagonal elements.

Starting with an initial guess $x^0 \in \mathbb{R}^n$, the iteration equation of the pointwise GS relaxation method applied to (2.10) is given by

$$(L + D)x^{k+1} = b - Ux^k \tag{2.12}$$

from which we obtain

$$(L + D)[x^{k+1} - x^k] = -U[x^k - x^{k-1}]$$

Hence

$$[x^{k+1} - x^k] = -(L + D)^{-1}U[x^k - x^{k-1}] \tag{2.13}$$

Therefore the GS relaxation iteration will converge for any given initial guess $x^0$ if and only if all eigenvalues of the matrix $(L + D)^{-1}U$ have magnitudes less than unity.

Now, applying the pointwise GJ relaxation to (2.10), we obtain the following iteration equation

$$Dx^{k+1} = b - (L + U)x^k \tag{2.14}$$

which leads to the following recursive error equation

$$x^{k+1} - x^k = -D^{-1}(L + U)(x^k - x^{k-1}) \tag{2.15}$$

Hence, the GJ relaxation iteration will converge for any given initial guess if and only if all eigenvalues of $D^{-1}(L + U)$ have magnitudes less than unity.

The convergence condition of the relaxation iteration clearly limits the class of systems to which relaxation can be applied. From practical points of view, it is also very important to be able to check whether or not relaxation can be applied before starting the iteration. This implies that we must find a numerical condition on the elements of the system to guarantee that the convergence condition is satisfied. For instance, if the matrix $A$ in (2.10) is strictly diagonally dominant [17], then the convergence condition of either GS or GJ relaxation

iteration will be satisfied. In circuit simulation, this sufficient condition must be further interpreted in terms of the properties of circuit elements. If (2.10) describes the "dc" node equations of a linear resistive circuit, then the condition that the circuit contains only resistors, i.e., there are no dependent sources, is sufficient to guarantee the strictly diagonally dominance of $A$. Obviously, this condition severely limits the type of linear circuits to which relaxation can be applied. Unfortunately weaker convergence conditions (although they exist) are difficult to characterize or compute. For this reason, relaxation decomposition has never been used in the "dc" simulation part of a circuit simulator.

The relaxation decomposition has been first used in the time domain circuit simulation by the *timing simulator* MOTIS [6]. This approach has later been modified and implemented in other *mixed-mode simulators* such as SPLICE [5], DIANA [9] and SAMSON [10]. The particular association of relaxation with the conventional numerical methods used by these simulators has given rise to a new area of time domain simulation called the *timing simulation*.

### 2.2.2.1. Timing Simulation.

Timing simulation is a time domain circuit simulation which uses a particular nonlinear relaxation approach for solving the nonlinear equations derived from the time discretization of the circuit differential equations. This type of simulation approach was originally introduced [6] for the simulation of MOS digital circuits. The particular characteristic of timing simulation is that the relaxation iteration is not carried out until convergence is achieved. Only one iteration (or sweep) of relaxation is performed and the results are accepted as the solutions of the nonlinear equations. Thus the timesteps must be kept small to reduce the inaccuracy of the solutions of the nonlinear equations. However, since the computational expense of taking one iteration is very small, the com-

puter time used in the timing simulation is usually much smaller than that of the standard simulator. In fact, with the inclusion of the *selective trace algorithm* or *event scheduling algorithm* in SPLICE [5] to exploit the latency of digital subcircuits, the timing simulation approach can be at least two order of magnitude faster than the standard simulation approach. Two critical assumptions that are responsible for the success of timing simulation are:

1) There is a grounded capacitor[6] to every node in the circuit.

2) The subcircuits to be decomposed have unidirectional or almost unidirectional properties both in the steady state and in the transient situations.

Unfortunately, there are many MOS digital circuits which contain large floating capacitors and/or trees of pass transistors (see Fig. 2.8). Experiments with these circuits have indicated that the timesteps have to be kept small in order to obtain accurate and reliable solutions. This is further complicated by the fact that there is no reliable technique to determine the appropriate sizes of these timesteps. The estimation of the local truncation error from the solutions in order to determine the timestep is no longer reliable since there is no guarantee that the nonlinear equations are accurately solved at every timepoint.

To illustrate the basic steps and numerical properties of timing simulation, we consider a circuit, such as the one shown in Fig. 2.8, whose node equations can be written as

$$C\dot{v} + f(v,u) = 0 \; ; \qquad v(0) = V \tag{2.16}$$

where $v(t) \in \mathbf{R}^n$ is the vector of the unknown node voltages, $u(t) \in \mathbf{R}^r$ is the vector of the independent sources, $C \in \mathbf{R}^{n \times n}$ is the node capacitance matrix in which $C_{ii}$ is the sum of the capacitances of all grounded and floating capacitors

[6] A grounded capacitor is a capacitor in which one of its terminals is connected to a known voltage source, such as an input voltage source or a constant voltage source, i.e., ground or power supply.

connected to the $i$-th node and $-C_{ij}, i \neq j$ is the total floating capacitance between the $i$-th and $j$-th nodes, and $f : \mathbf{R}^n \times \mathbf{R}^r \rightarrow \mathbf{R}^n$ is a Lipschitz continuous function [25] each component of which represents the sum of currents feeding the capacitors at the $i$-th node. Note that all capacitors are assumed to be linear and that $C$ is strictly diagonally dominant since there is a grounded capacitor to every node.

In timing simulation, the time derivative $\dot{v}$ is discretized by an implicit integration formula such as the Backward Euler formula in MOTIS and SPLICE or the Trapezoidal formula in MOTIS-C [27]. For this example, we assume that fixed timesteps of size $h$ are used and that the time derivative is discretized by the Backward Euler formula, i.e.,

$$\dot{v}(t_{i+1}) \approx \frac{1}{h}(v(t_{i+1}) - v(t_i))$$

Hence, the nonlinear equations obtained through the discretization of (2.16) are given by

$$C(v_{i+1} - v_i) + hf(v_{i+1}, u_{i+1}) = 0 \qquad (2.17)$$

where $v_{i+1} \approx v(t_{i+1})$, $v_i \approx v(t_i)$ and $u_{i+1} = u(t_{i+1})$. If (2.17) is solved exactly, then the sequence of $v_i$ will possess all the numerical properties, i.e., consistency and stability, of the Backward Euler integration method. This is, of course, not the case in timing simulation. Let

$$C = L + D + U \qquad (2.18)$$

where $L \in \mathbf{R}^{n \times n}$ is a strictly lower triangular matrix, $D \in \mathbf{R}^{n \times n}$ is a diagonal matrix and $U \in \mathbf{R}^{n \times n}$ is a strictly upper triangular matrix. In timing simulation, (2.17) can be solved either by GJ relaxation as in MOTIS or by GS relaxation as in MOTIS-C and SPLICE but only one iteration of relaxation is performed. Applying one iteration of the pointwise GJ relaxation to (2.17), we obtain the following equations

$$D(v_{i+1}^{GJ} - v_i^{GJ}) + (L + U)(v_{i+1}^{GJ^0} - v_i^{GJ}) + hf_{GJ}(v_{i+1}^{GJ}, v_{i+1}^{GJ^0}, u_{i+1}) = 0 \qquad (2.19)$$

where $v_{i+1}^{GJ^0} \in \mathbf{R}^n$ is the guess for the relaxation and, for each component index $j = 1,2,\ldots,n$,

$$f_{GJ_j}(v_{i+1}^{GJ}, v_{i+1}^{GJ^0}, u_{i+1}) \triangleq f_j(v_{i+1_1}^{GJ^0}, \ldots, v_{i+1_{j-1}}^{GJ^0}, v_{i+1_j}^{GJ}, v_{i+1_{j+1}}^{GJ^0}, \ldots, v_{i+1_n}^{GJ^0}, u_{i+1}) \qquad (2.20)$$

Similarly, applying one iteration of the pointwise GS relaxation to (2.17), we obtain the following equations

$$(D + L)(v_{i+1}^{GS} - v_i^{GS}) + U(v_{i+1}^{GS^0} - v_i^{GS}) + hf_{GS}(v_{i+1}^{GS}, v_{i+1}^{GS^0}, u_{i+1}) = 0 \qquad (2.21)$$

where $v_{i+1}^{GS^0} \in \mathbf{R}^n$ is the guess for the relaxation and, for each component index $j = 1,2,\ldots,n$,

$$f_{GS_j}(v_{i+1}^{GS}, v_{i+1}^{GS^0}, u_{i+1}) \triangleq f_j(v_{i+1_1}^{GS}, \ldots, v_{i+1_j}^{GS}, v_{i+1_{j+1}}^{GS^0}, \ldots, v_{i+1_n}^{GS^0}, u_{i+1}) \qquad (2.22)$$

Note that neither (2.19) nor (2.21) are equivalent to (2.17). Hence neither the sequence of $v_i^{GS}$ nor the sequence of $v_i^{GJ}$ necessarily possess the same numerical properties as the sequence of $v_i$. In other words, the numerical properties of the Backward Euler integration method are not necessarily preserved through the one sweep of the relaxation process. Therefore, a complete analysis of the numerical properties of these *combined integration-relaxation* methods has to be carried out to characterize them. Such an analysis has been done in [18] for the case when $v_{i+1}^{GS^0} = v_i^{GS}$ and $v_{i+1}^{GJ^0} = v_i^{GJ}$. It is interesting to note that in this case, when $C$ is not diagonal, the combined integration-relaxation methods are not even consistent, i.e., the sequence of $v_i^{GS}$ or $v_i^{GJ}$ does not converge to the true solution of the original differential equations (2.16) as the stepsize $h$ goes to zero. This result can be easily shown by examining (2.19) and (2.21) when $v_{i+1}^{GS^0} = v_i^{GS}$ and $v_{i+1}^{GJ^0} = v_i^{GJ}$. From (2.19) we obtain

$$D(v_{i+1}^{GJ} - v_i^{GJ}) + hf_{GJ}(v_{i+1}^{GJ}, v_i^{GJ}, u_{i+1}) = 0 \qquad (2.23)$$

and from (2.21) we obtain

$$(D + L)(v_{i+1}^{GS} - v_i^{GS}) + hf_{GS}(v_{i+1}^{GS}, v_i^{GS}, u_{i+1}) = 0 \qquad (2.24)$$

We immediately see that the effects of $L$ and $U$ which are due to the floating capacitors completely disappear from (2.23) and partially disappear from (2.24). In fact (2.23) can be exactly obtained by applying the combined integration-GJ-relaxation to the following differential equations (with the initial guess $v_{i+1}^{GJ^0} = v_i^{GJ}$)

$$D\dot{v} + f(v, u) = 0 \; ; \qquad v(0) = V \qquad (2.25)$$

and similarly (2.24) can be exactly obtained by applying the combined integration-GS-relaxation to the following differential equations (with the initial guess $v_{i+1}^{GS^0} = v_i^{GS}$)

$$(D + L)\dot{v} + f(v, u) = 0 \; ; \qquad v(0) = V \qquad (2.26)$$

That is, these methods are solving dynamical systems which are not the same as the original system described by (2.16). The circuit interpretations of both (2.25) and (2.26) for the original circuit of Fig. 2.8 are shown in Fig. 2.9 and Fig. 2.10 respectively. This is a good example to show why these methods work rather well when there are no floating capacitors or when the floating capacitances are small compared to the grounded capacitances.

Some of the drawbacks of the above methods can be overcome. It can be easily shown that the use of the Forward Euler formula to generate the initial guess for the relaxation will at least make the combined integration-relaxation method consistent with the circuit equations. Also the study carried out in [18] has indicated that the use of another type of relaxation based on an idea by Kahan [19] results in a class of combined integration-relaxation methods, called the *modified symmetric Gauss-Seidel* integration, which has better numerical

properties. Another simple way to improve the reliability of timing simulation is to continue the relaxation iteration until convergence is achieved. This latter technique is dicussed in more detail in Appendix B.

### 2.2.3. Concluding Remarks.

We have described and classified various decomposition techniques that have been proposed and implemented. Whereas the relaxation approach to solving linear and nonlinear algebraic equations has been treated quite extensively (see [17] for the linear case and [20] for the nonlinear case), the study of the relaxation approach at the differential equation level is still open both as a numerical method and as a new tool for performing time domain simulation. At this level, each decomposed subsystem is still a system of differential equations and hence can be solved in the time domain by using conventional numerical methods, e.g. the Backward Euler formula, the Newton-Raphson method and the Gaussian Elimination method as shown in Fig. 2.7. The purpose of this dissertation is to provide a complete study of this new decomposition technique which we call the *Waveform Relaxation* (WR) method.

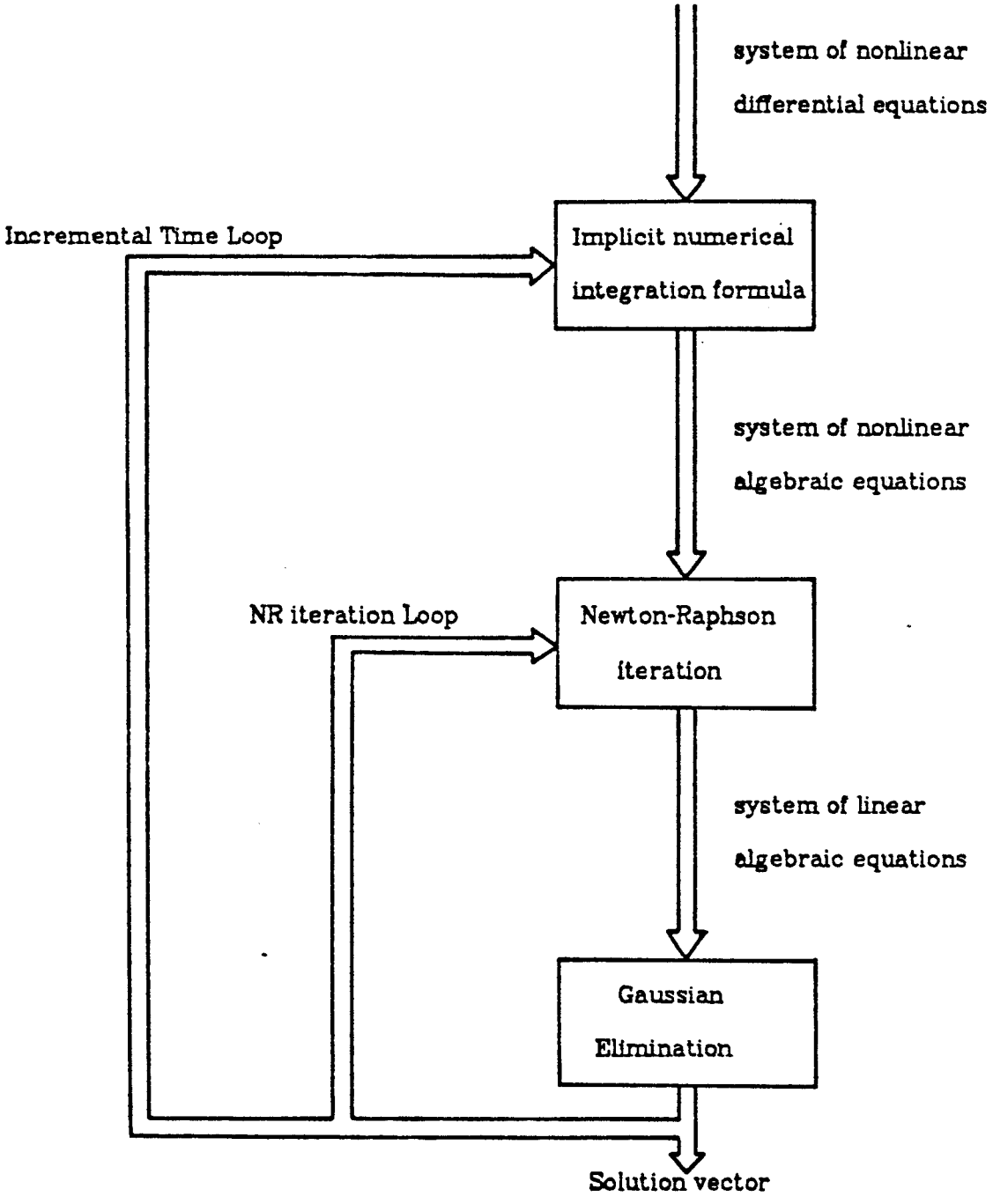



**Fig. 2.1**

Hierachical organization of conventional numerical methods

for time domain simulation

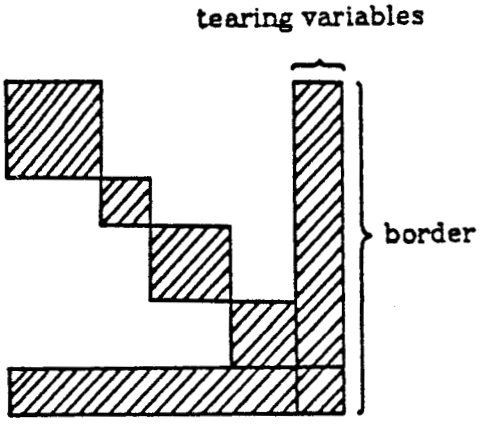



**Fig. 2.2a**

Bordered Block Diagonal (BBD) form of a matrix.

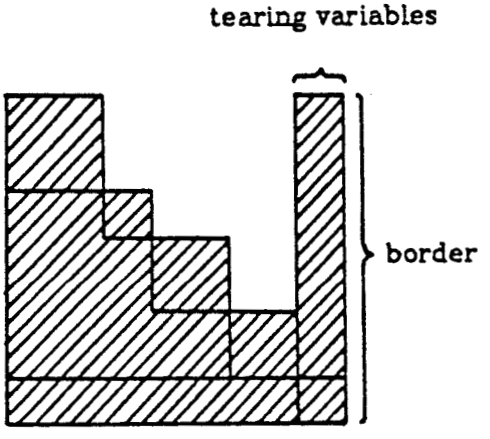



**Fig. 2.2b**

Bordered Block Lower Triangular (BBLT) form of a matrix.

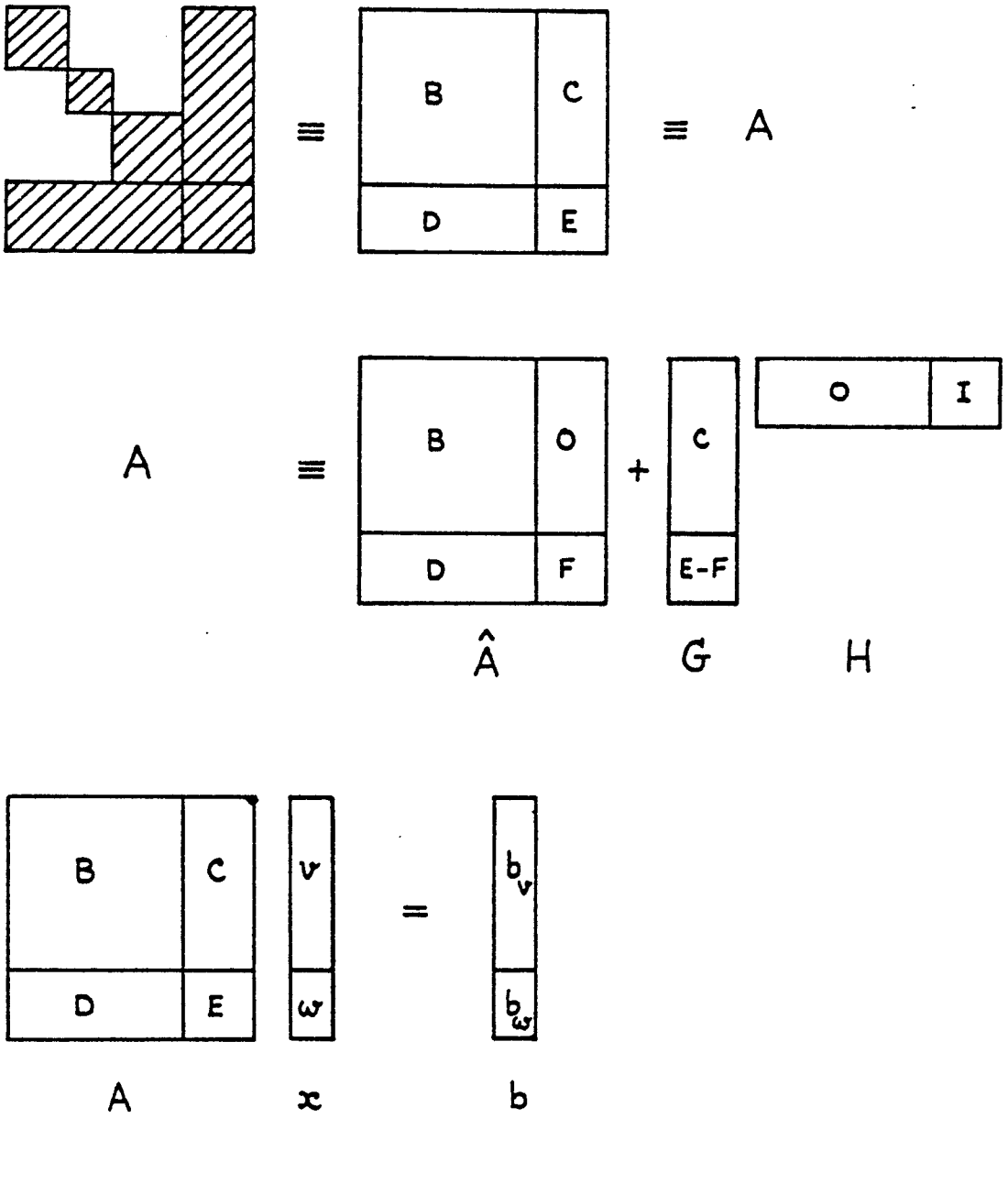



**Fig. 2.3**

Various terms associated with Block LU Factorization and Tearing Algorithm.

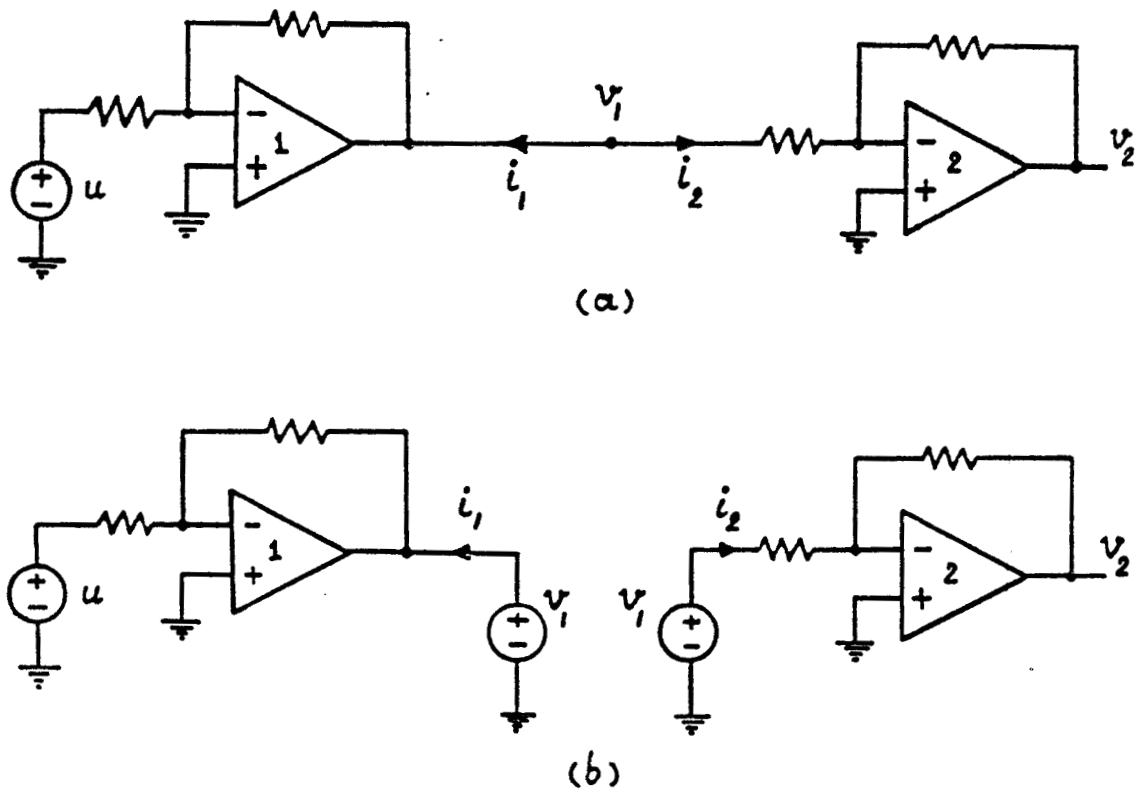



**Fig. 2.4**

**a)** An interconnection of two analog inverters.

**b)** Node tearing decomposition of the circuit in Fig. 2.4a.

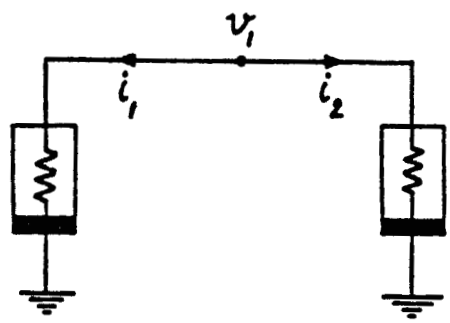
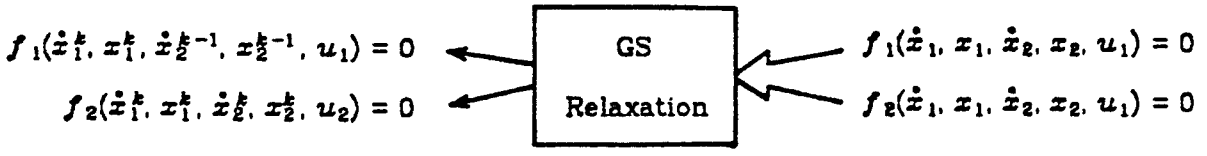


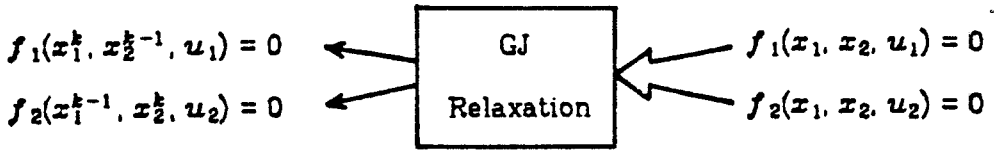

**Fig. 2.5**

The reduced circuit of the circuit in

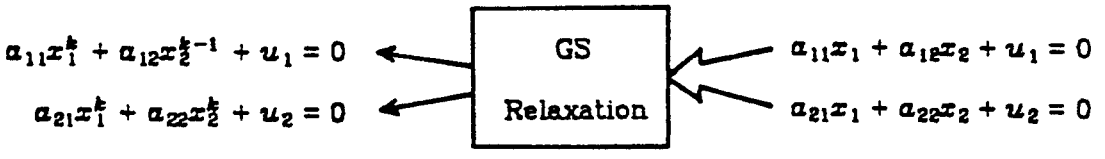
Fig. 2.4a as viewed by the tearing variable $v_1$.




a) Relaxation decomposition of differential equations




b) Relaxation decomposition of nonlinear algebraic equations




c) Relaxation decomposition of linear algebraic equations

Fig. 2.6

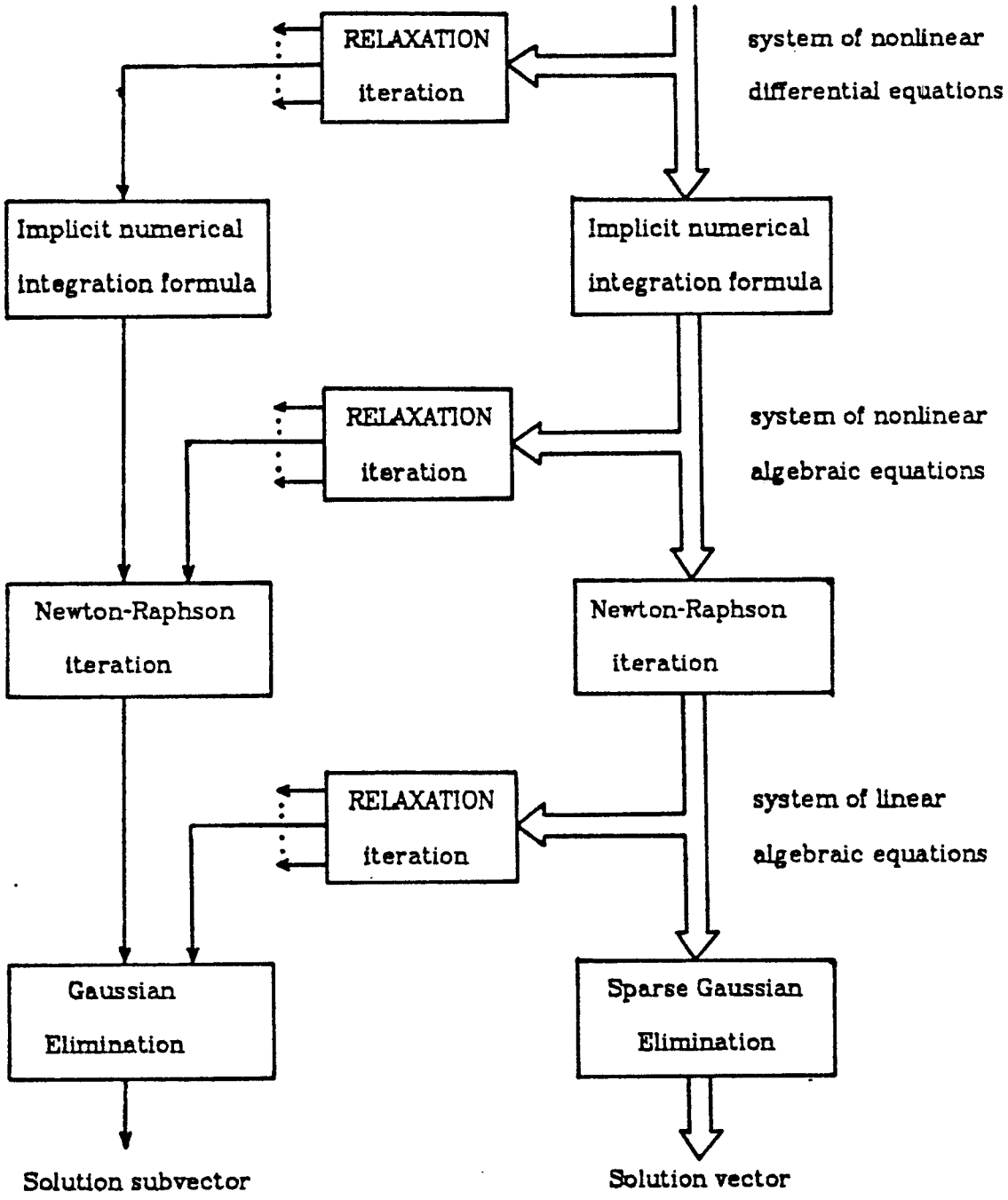



Fig. 2.7

The use of relaxation at various levels of system of equations.

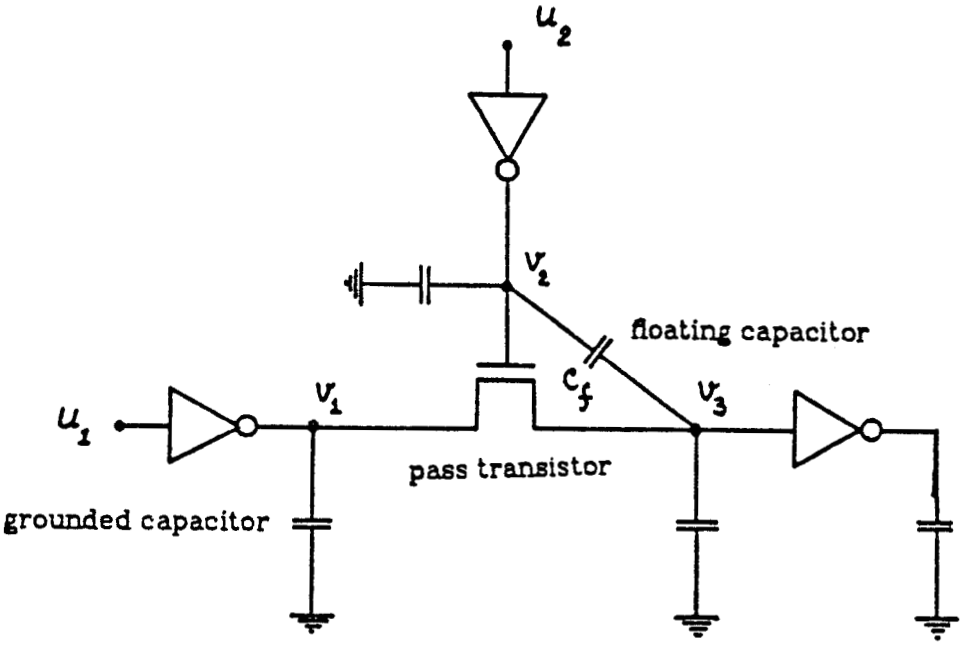



**Fig. 2.8**

A typical MOS circuit that contains a pass transistor and floating capacitors

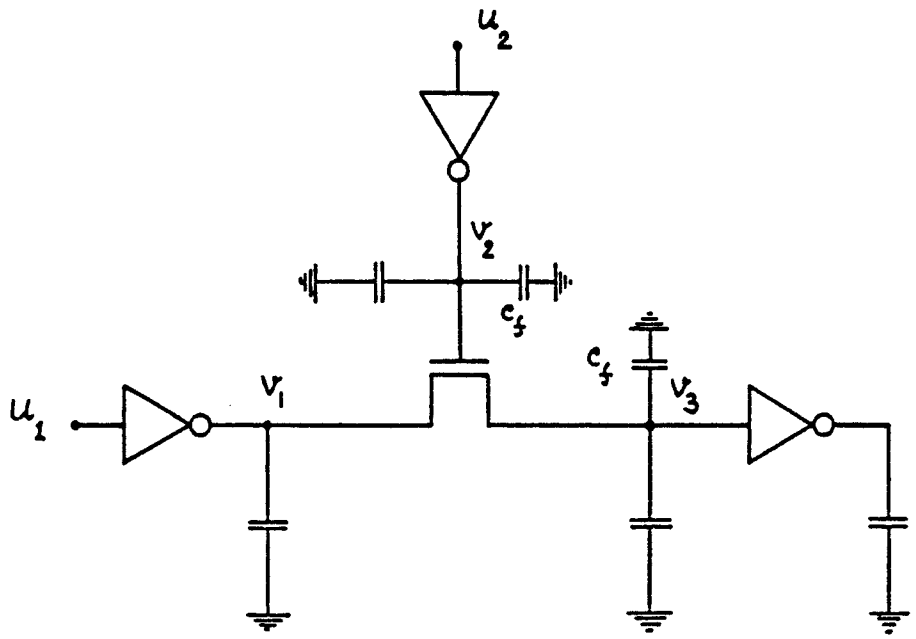



**Fig. 2.9**

The circuit interpretation of the application of the combined integration-GJ-relaxation to the circuit of Fig. 2.8 ( according to equation (2.25) ).

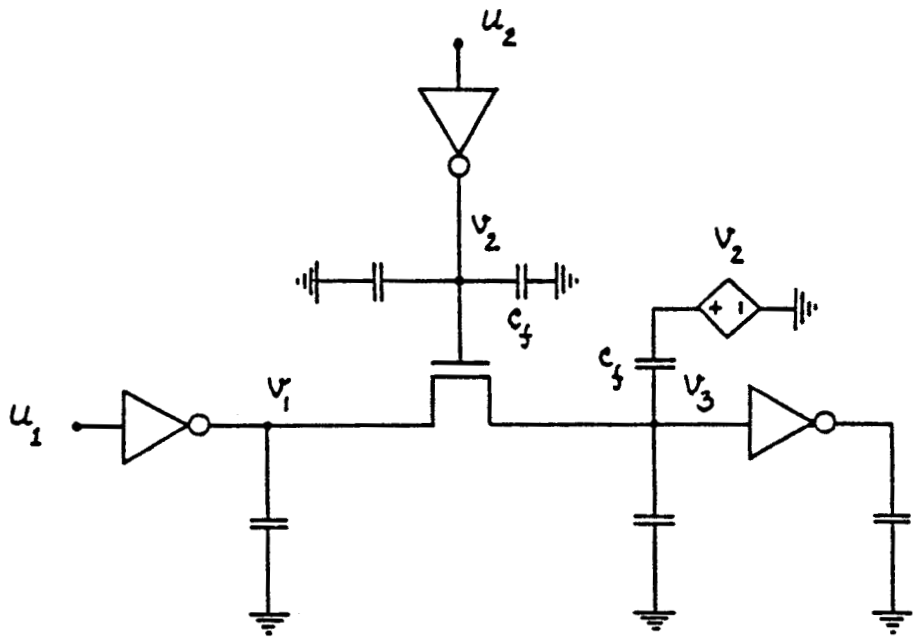



**Fig. 2.10**

The circuit interpretation of the application of the combined integration-GS-relaxation to the circuit of Fig. 2.8 ( according to equation (2.26) ).

# Chapter 3

# The Waveform Relaxation Method

In this chapter we describe the basic mathematical concept of the Waveform Relaxation (WR) method together with a few circuit examples to demonstrate the physical interpretation of the decomposition achieved by the method.

## 3.1. Mathematical Formulation.

We consider dynamical systems which can be described by a system of mixed implicit algebraic-differential equations of the form:

$$F(\dot{y}, y, u) = 0 \tag{3.1a}$$

$$E(y(0) - y_0) = 0 \tag{3.1b}$$

where $y(t) \in \mathbb{R}^p$ is the vector of the unknown variables at time $t$, $\dot{y}(t) \in \mathbb{R}^p$ is the time derivative of $y$ at time $t$, $u(t) \in \mathbb{R}^r$ is the vector of the input variables at time $t$, $y_0 \in \mathbb{R}^p$ is the given initial value of $y$, $F : \mathbb{R}^p \times \mathbb{R}^p \times \mathbb{R}^r \to \mathbb{R}^p$ is a continuous function, and $E \in \mathbb{R}^{n \times p}$, $n \leq p$ is a matrix of rank $n$ such that $Ey(t)$ is the state of the system at time $t$.

Note that equation (3.1b) is meant to supply the initial conditions for the state variables [23] of (3.1a). We shall assume that $y_0$ is chosen so as to give $y(0) = y_0$, i.e., $y_0$ also satisfies all the algebraic relations embedded in (3.1a). In circuit simulation, $y_0$ is usually obtained from the so called "dc" solution of the system, i.e., it satisfies

$$F(\dot{y}(0), y_0, u(0)) = 0 \; ; \quad \dot{y}(0) = 0 \tag{3.2}$$

The general structure of a WR algorithm for analyzing (3.1) over a given time interval $[0,T]$ consists of two major processes, namely the *assignment-partitioning* process and the *relaxation* process.

## 3.2. The Assignment-Partitioning Process.

In the assignment-partitioning process, each unknown variable is assigned to an equation of (3.1a) in which it is involved. However, no two variables can be assigned to the same equation. Then (3.1a) is partitioned into $m$ disjoint[1] subsystems of equations, each of which may have only differential equations or only algebraic equations or both. Without loss of generality, we can rewrite (3.1) after being processed by the assignment-partitioning process as follows:

$$\begin{bmatrix} F_1(\dot{y}_1, y_1, d_1, u) \\ \vdots \\ F_m(\dot{y}_m, y_m, d_m, u) \end{bmatrix} = 0 \tag{3.3a}$$

$$E(y(0) - y_0) = 0 \tag{3.3b}$$

where, for each $i = 1,2,\ldots,m$, $y_i \in \mathbb{R}^{p_i}$ is the subvector of the unknown variables assigned to the $i$-th partitioned subsystem, $F_i : \mathbb{R}^{p_i} \times \mathbb{R}^{p_i} \times \mathbb{R}^{2p-2p_i} \times \mathbb{R}^r \to \mathbb{R}^{p_i}$ is a continuous function, and

$$d_i \triangleq \text{col}^2(y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_m, \dot{y}_1, \ldots, \dot{y}_{i-1}, \dot{y}_{i+1}, \ldots, \dot{y}_m) \tag{3.3c}$$

It is clear that if the vectors $d_i$, $i = 1,2,\ldots,m$, are treated as the input variables of the system described by (3.3a), then the system can be easily solved by solving $m$ independent subsystems associated with $F_1, F_2, \ldots, F_m$ respectively. Therefore they are called the *decoupling* vectors of the subsystems. This gives rise to the notion of the decomposed system as given in the following definition.

---

[1] There are cases in which the algorithm has better convergence properties if the subsystems are nondisjoint. For such cases, we can consider the nondisjoint subsystems as being obtained from partitioning an augmented system of equations with an augmented set of unknown variables.

[2] $\text{col}(a, b) \triangleq \begin{bmatrix} a \\ b \end{bmatrix}$

**Definition 3.1** The *decomposed system* associated with an assignment-partitioning process applied to (3.1) consists of $m$ independent subsystems, called *decomposed subsystems*, each of which is described by

$$F_i(\dot{y}_i, y_i, \tilde{u}_i, u) = 0 \tag{3.4a}$$

$$E_i(y_i(0) - y_{0i}) = 0 \tag{3.4b}$$

where $y_{0i} \in \mathbb{R}^{p_i}$ is the subvector of the given initial vector $y_0$, $\tilde{u}_i \in \mathbb{R}^{2p-2p_i}$ is the vector of the decoupling inputs, $F_i : \mathbb{R}^{p_i} \times \mathbb{R}^{p_i} \times \mathbb{R}^{2p-2p_i} \times \mathbb{R}^r \to \mathbb{R}^{p_i}$ is a continuous function as given by (3.3a), and $E_i \in \mathbb{R}^{n_i \times p_i}$, $n_i \leq p_i$ is a matrix of rank $n_i$ such that $E_i y_i$ is a state vector of the $i$-th decomposed subsystem described by (3.4).

■

## 3.3. The Relaxation Process.

The relaxation process is an iterative process. For simplicity, we shall consider two most commonly used types of relaxation, namely the *Gauss-Seidel* [17] (GS) relaxation and the *Gauss-Jacobi* [17] (GJ) relaxation. The relaxation process starts with an initial guess of the waveform solution of the original dynamical equations (3.3) in order to initialize the approximate waveforms of the decoupling vectors. During each iteration, each decomposed subsystem is solved for its assigned variables in the given time interval $[0, T]$ by using the approximate waveform of its decoupling vector. For the GS relaxation, the waveform solution obtained by solving one decomposed subsystem is immediately used to update the approximate waveforms of the decoupling vectors of the other subsystems. For the GJ relaxation, all waveforms of the decoupling vectors are updated at the beginning of the next iteration. The relaxation process is carried out repeatedly until satisfactory convergence is achieved.

Let the superscript index $k$ denote the WR iteration count. Then the general structure of a WR algorithm can be formally described as follows:

**The WR Algorithm Model 3.1**

Step 0: (Assignment-partitioning process)

Assign the unknown variables to the equations in (3.1) and partition (3.1) into $m$ subsystems of equations as given by (3.3).

Step 1: (Initialization of the relaxation process)

Set $k = 1$ and guess an initial waveform $(y^0(t) ; t \in [0,T])$ such that $y^0(0) = y(0) = y_0$.

Step 2: (Analyzing the decomposed system at the $k$-th WR iteration)

For each $i = 1,2,\ldots,m$, set

$$d_i^k = \text{col}\,(y_1^k, \ldots, y_{i-1}^k, y_{i+1}^{k-1}, \ldots, y_m^{k-1}, \dot{y}_1^k, \ldots, \dot{y}_{i-1}^k, \dot{y}_{i+1}^{k-1}, \ldots, \dot{y}_m^{k-1})$$

for the GS relaxation, or

$$d_i^k = \text{col}\,(y_1^{k-1}, \ldots, y_{i-1}^{k-1}, y_{i+1}^{k-1}, \ldots, y_m^{k-1}, \dot{y}_1^{k-1}, \ldots, \dot{y}_{i-1}^{k-1}, \dot{y}_{i+1}^{k-1}, \ldots, \dot{y}_m^{k-1})$$

for the GJ relaxation, and solve for $(y_i^k(t) ; t \in [0,T])$ from

$$F_i(\dot{y}_i^k, y_i^k, d_i^k, u) = 0 \qquad (3.5a)$$

$$E_i(y_i^k(0) - y_i(0)) = 0 \qquad (3.5b)$$

Step 3: (Iteration)

Set $k = k+1$ and go to step 2. ∎

**Remarks.**

1) A simple guess for $(y^0(t) ; t \in [0,T])$ is $y^0(t) = y(0)$ for all $t \in [0,T]$.

2) In the actual implementation, the relaxation iteration will stop when the difference between the waveforms $(y^k(t) ; t \in [0,T])$ and $(y^{k-1}(t) ; t \in [0,T])$, i.e., $\max_{t \in [0,T]} \|y^k(t) - y^{k-1}(t)\|$, is sufficiently small.

3) In analogy to the classical relaxation methods for solving linear or nonlinear algebraic equations [17,20], it is possible to modify a WR algorithm by using a *relaxation parameter* $\omega \in (0,2)$. With $\omega$, the iteration equation (3.5) is modified to yield

$$F_i(\dot{\tilde{y}}_i^k, \tilde{y}_i^k, d_i^k, u) = 0 \tag{3.6a}$$

$$E_i(\tilde{y}_i^k(0) - y_i(0)) = 0 \tag{3.6b}$$

$$y_i^k = y_i^{k-1} + \omega(\tilde{y}_i^k - y_i^{k-1}) \tag{3.6c}$$

4) Note the following two important characteristics of the WR Algorithm Model 3.1.

   a) The analysis of the original system is decomposed into the independent analysis of $m$ subsystems.

   b) The relaxation process is carried out on the entire waveforms, i.e. during each iteration each subsystem is individually analyzed for the entire given time interval $[0,T]$. ■

## 3.4. Circuit Examples and Their Physical Interpretations.

In this section, we shall use a few specific examples to demonstrate the applications of the WR Algorithm Model 3.1 in the analysis of lumped dynamical circuits and to give the circuit interpretation of the decompostion. Different formulations of the circuit equations will be used to illustrate the resulting decompositions.

**Example 3.1**

Consider the circuit shown in Fig. 3.1. Using Nodal Analysis [23] formulation with $v_1$ and $v_2$ as the circuit variables, the node equations of the circuit are

$$(C_1 + C_3)\dot{v}_1 - C_3\dot{v}_2 + G_1 v_1 = J \; ; \qquad v_1(0) = V_1 \tag{3.7a}$$

$$(C_2 + C_3)\dot{v}_2 - C_3\dot{v}_1 + G_2 v_2 = 0 \; ; \qquad v_2(0) = V_2 \tag{3.7b}$$

Let $v_1$ and $v_2$ be assigned to (3.7a) and (3.7b) respectively and let (3.7) be partitioned into two subsystems consisting of {(3.7a)} and {(3.7b)}. Applying the WR Algorithm Model 3.1, the $k$-th iteration of the corresponding GS-WR algorithm corresponds to solving

$$(C_1 + C_3)\dot{v}_1^k - C_3\dot{v}_2^{k-1} + G_1 v_1^k = J \; ; \qquad v_1^k(0) = V_1 \tag{3.8a}$$

for the first subsystem, and

$$(C_2 + C_3)\dot{v}_2^k - C_3\dot{v}_1^k + G_2 v_2^k = 0 \; ; \qquad v_2^k(0) = V_2 \tag{3.8b}$$

for the second subsystem. The circuit interpretation of the decomposed circuit at the $k$-th iteration, as described by (3.8), is shown in Fig. 3.2. ∎

**Example 3.2**

Consider the circuit shown in Fig. 3.1. Using Modified Nodal Analysis [21] formulation with $v_1$, $v_2$ and $i_3$ as the circuit variables, the circuit equations can be written as

$$C_1\dot{v}_1 + G_1 v_1 + i_3 = J \; ; \qquad v_1(0) = V_1 \tag{3.9a}$$

$$C_2\dot{v}_2 + G_2 v_2 - i_3 = 0 \; ; \qquad v_2(0) = V_2 \tag{3.9b}$$

$$i_3 - C_3(\dot{v}_1 - \dot{v}_2) = 0 \tag{3.9c}$$

Let $v_1$, $v_2$ and $i_3$ be assigned to (3.9a), (3.9b) and (3.9c) respectively and let (3.9) be partitioned into two subsystems consisting of {(3.9a)} and {(3.9b),(3.9c)}. Applying the WR Algorithm Model 3.1, the $k$-th iteration of the resulting GJ-WR algorithm corresponds to solving

$$C_1\dot{v}_1^k + G_1 v_1^k + i_3^{k-1} = J \; ; \qquad v_1^k(0) = V_1 \tag{3.10a}$$

for the first subsystem, and

$$C_2\dot{v}_2^k + G_2 v_2^k - i_3^k = 0 \; ; \qquad v_2^k(0) = V_2 \tag{3.10b}$$

$$i_3^k - C_3(\dot{v}_1^{k-1} - \dot{v}_2^k) = 0 \qquad (3.10c)$$

for the second subsystem. The circuit interpretation of the decomposed circuit at the $k$-th iteration, as described by (3.10), is shown in Fig. 3.3. ■

**Example 3.3**

Consider again the circuit shown in Fig. 3.1. Using a "Sparse Tableau [22] like" formulation with $v_1$, $v_2$, $v_3$, $i_1$, $i_2$ and $i_3$ as circuit variables, the circuit equations can be written as

$$C_1\dot{v}_1 - i_1 = 0 \; ; \qquad v_1(0) = V_1 \qquad (3.11a)$$

$$C_2\dot{v}_2 - i_2 = 0 \; ; \qquad v_2(0) = V_2 \qquad (3.11b)$$

$$C_3\dot{v}_3 - i_3 = 0 \qquad (3.11c)$$

$$v_3 - v_1 + v_2 = 0 \qquad (3.11d)$$

$$G_1v_1 + i_1 + i_3 = J \qquad (3.11e)$$

$$G_2v_2 + i_2 - i_3 = 0 \qquad (3.11f)$$

Let $v_1$, $v_2$, $i_3$, $v_3$, $i_1$, $i_2$ be assigned to (3.11a) through (3.11f) respectively and let the system be partitioned into three subsystems consisting of {(3.11a),(3.11e)}, {(3.11b),(3.11f)} and {(3.11c),(3.11d)}. Note that we cannot assign $v_1$, $v_2$, $v_3$ to (3.11a), (3.11b), (3.11c) respectively since one of them has to be assigned to (3.11d). Applying the WR Algorithm Model 3.1, the $k$-th iteration of the resulting GJ-WR algorithm corresponds to solving

$$C_1\dot{v}_1^k - i_1^k = 0 \; ; \qquad v_1^k(0) = V_1 \qquad (3.12a)$$

$$G_1v_1^k + i_1^k + i_3^{k-1} = J \qquad (3.12b)$$

for the first subsystem,

$$C_2\dot{v}_2^k - i_2^k = 0 \; ; \qquad v_2^k(0) = V_2 \qquad (3.12c)$$

$$G_2v_2^k + i_2^k - i_3^{k-1} = 0 \qquad (3.12d)$$

for the second subsystem, and

$$C_3\dot{v}_3^k - i_3^k = 0 \qquad (3.12e)$$

$$v_3^k - v_1^{k-1} + v_2^{k-1} = 0 \qquad (3.12f)$$

for the third subsystem. The circuit interpretation of the decomposed system at the $k$-th iteration, as described by (3.12), is shown in Fig. 3.4. ∎

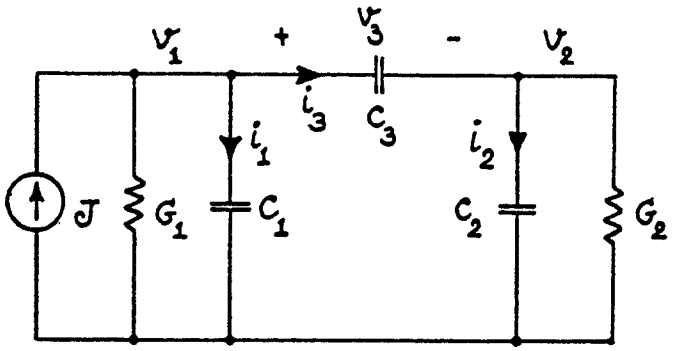



Fig. 3.1

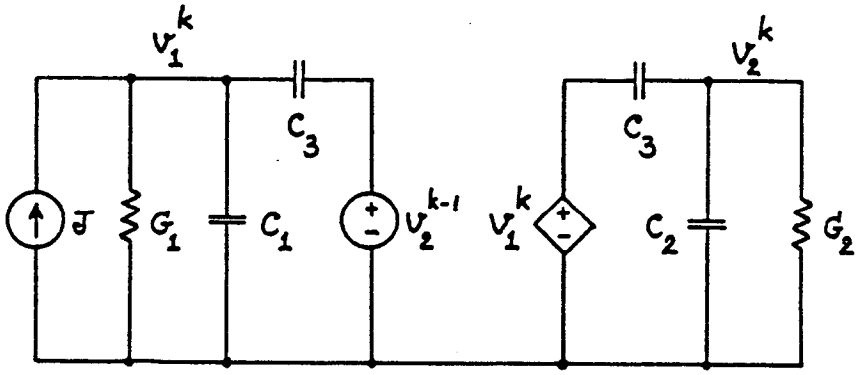



Fig. 3.2

Example 3.1: Circuit interpretation of a GS-WR algorithm applied to the circuit of Fig. 3.1.

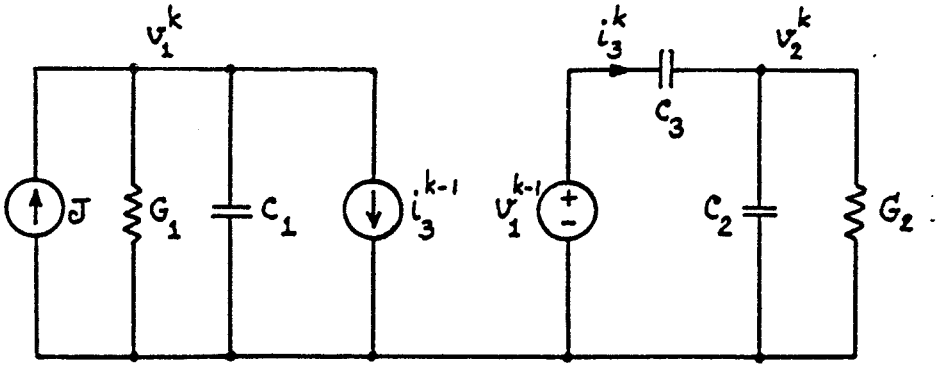



**Fig. 3.3**

Example 3.2: Circuit interpretation of a GJ-WR algorithm applied to the circuit of Fig. 3.1.

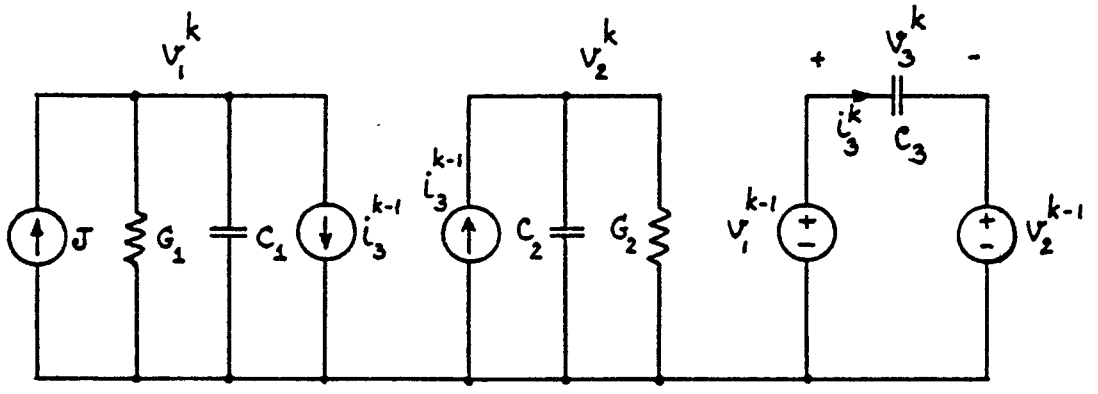



**Fig. 3.4**

Example 3.3: Circuit interpretation of a GJ-WR algorithm applied to the circuit of Fig. 3.1.

# Chapter 4

# Consistency of the Assignment-Partitioning (AP) Process

In this chapter we introduce the concept of *consistency* of the assignment-partitioning (AP) process and show how an inconsistent AP process can lead to serious convergence problems for the relaxation process of the WR algorithm. The formal approach for finding a consistent AP process or verifying its consistency directly from the system equations will be addressed by using techniques based on the graph-theoretic interpretation of the *algebraic-differential dependency matrix* associated with the system equations.

## 4.1. Definition of Consistency and Examples.

Decomposition of a system of equations into subsystems of equations through relaxation is specified by the AP process. If the system is purely algebraic, i.e., it contains only algebraic equations, then the decomposed system as defined in the previous chapter will also be purely algebraic independent of the choice of assignment and partitioning. However, in our case the given system contains differential equations. Hence, it is possible that, for some particular choices of assignment and partitioning, some differential equations of the system are converted into algebraic equations in the decomposed system.

To show the effect of the AP process on the dynamical behaviour of its associated decomposed system, consider the following system of equations

$$\dot{x}_1 + x_1 + x_2 + u_1 = 0 \qquad (4.1a)$$

$$\dot{x}_2 + x_2 + x_1 + u_2 = 0 \qquad (4.1b)$$

Assume that we want to partition this system into 2 subsystems consisting of {(4.1a)} and {(4.1b)}. If we choose to assign $x_1$ and $x_2$ to (4.1a) and (4.1b) respectively, then the decomposed system according to Definition 3.1 is given by

$$\begin{aligned} \dot{x}_1 + x_1 + \tilde{u}_1 + u_1 &= 0 \\ \dot{x}_2 + x_2 + \tilde{u}_2 + u_2 &= 0 \end{aligned}$$

which is a dynamical system with two state variables $x_1$ and $x_2$ as in the original system (4.1). On the other hand, if we choose to assign $x_1$ to (4.1b) and $x_2$ to (4.1a), then the decomposed system is given by

$$\begin{aligned} x_2 + \tilde{u}_1 + \dot{\tilde{u}}_1 + u_1 &= 0 \\ x_1 + \tilde{u}_2 + \dot{\tilde{u}}_2 + u_2 &= 0 \end{aligned}$$

which is a purely algebraic system, i.e., it has no state variable.

From the above example, it is clear that different choices of the AP process can result in decomposed systems with entirely different dynamical behaviours. Furthermore, it is very important to choose an AP process such that the dynamical behaviour of its associated decomposed system is as close to that of the original system as possible in order to obtain good convergence properties of the relaxation process. Therefore, by using the concept of *state variables* [23], we can classify the AP processes into two categories, namely the *consistent* AP process and the *inconsistent* AP process.

**Definition 4.1** An AP process is said to be *consistent* with a given dynamical system if any choice[1] of the state vector of its associated decomposed system is also a valid choice of the state vector of the given system. The decomposed system associated with a consistent AP process is also said to be consistent with the given system and a WR algorithm that uses a consistent AP process is called a *consistent WR algorithm.* ■

[1] In general, the choice of state variables of a dynamical system is not unique.

The following two examples illustrate why consistency of the AP process plays an important role in the convergence of the relaxation process.

**Example 4.1** Consider a dynamical system described by

$$\dot{y}_1 + y_2 - u = 0 \; ; \qquad y_1(0) = 0 \tag{4.2a}$$

$$y_1 - y_2 = 0 \tag{4.2b}$$

This system has one state variable. Suppose that $y_1$ and $y_2$ are assigned to (4.2b) and (4.2a) respectively and that the system is partitioned into two subsystems consisting of {(4.2a)} and {(4.2b)}. Applying the WR Algorithm Model 3.1, the $k$-th iteration of the resulting GS-WR algorithm corresponds to solving

$$y_2^k = u - \dot{y}_1^{k-1} \tag{4.3a}$$

$$y_1^k = y_2^k \tag{4.3b}$$

Notice that the decomposed system at the $k$-th iteration, as described by (4.3), is purely algebraic. Hence, this AP process is inconsistent with the given system. From (4.3), it is easy to derive that the iterated solution of the decomposed system is given by

$$y_1^k(t) = y_2^k(t) = \sum_{j=0}^{k-1}(-1)^j \frac{d^j}{dt_j}u(t) + (-1)^k \frac{d^k}{dt^k}y_1^0(t)$$

which leads to the following results:

a) If the initial guess of the relaxation process is $y_1^0(t) = e^{-at}$ with $a > 0$, then the iterated solution $y_1^k(\cdot)$ or $y_2^k(\cdot)$ will diverge. This result is independent of the input $u(\cdot)$.

b) If the input is $u(t) = e^{-at}$ with $a > 1$ and $y_1^0(t) = 0$, then the iterated solution will diverge.

c) If the input is piecewise continuous with at least one discontinuity, e.g.

$$u(t) = \begin{cases} 0 & t \le 1 \\ 1 & t > 1 \end{cases}$$

and $y_1^0(t) = 0$, then the iterated solution will be discontinuous and unbounded at the points of discontinuity of the input whereas the exact solution of the given system is continuous and bounded within any finite time interval.

d) If $u(t) = a$ and $y_1^0(t) = 0$, then the iterated solution $y_1^k(t)$ or $y_2^k(t)$ converges to $\hat{y}_1(t) = \hat{y}_2(t) = a$ whereas the exact solution of the given system is $y_1(t) = y_2(t) = a(1-e^{-t})$. ∎

**Example 4.2** Consider again the same dynamical system described by (4.2). This time we assign $y_1$ to (4.2a) and $y_2$ to (4.2b). Applying the WR Algorithm Model 3.1, the $k$-th iteration of the resulting GS-WR algorithm corresponds to solving

$$\dot{y}_1^k = u - y_2^{k-1} \; ; \qquad y_1^k(0) = 0 \tag{4.4a}$$

$$y_2^k = y_1^k \tag{4.4b}$$

Notice that this time the decomposed system at the $k$-th iteration, as described by (4.4), has one state variable $y_1$ which is also a state variable of the given system. Hence, this AP process is consistent with the given system of equations (4.2). In contrast to Example 4.1, we shall show that, for any given time interval $[0,T]$, the iterated solution of (4.4) always converges to the exact solution of the given system (4.2) independent of the initial guess $y_1^0(\cdot)$ and the input $u(\cdot)$.

From (4.4) and (4.2), we obtain

$$\dot{y}_1^k - \dot{y}_1 = -(y_1^{k-1} - y_1) \; ; \qquad y_1^k(0) - y_1(0) = 0$$

from which the solution is

$$y_1^k(t) - y_1(t) = -\int_0^t [y_1^{k-1}(\tau) - y_1(\tau)]d\tau \tag{4.5}$$

Multiplying (4.5) by $e^{-2t}$, we have

$$e^{-2t}(y_1^k(t) - y_1(t)) = -e^{-2t}\int_0^t e^{2\tau}[e^{-2\tau}(y_1^{k-1}(\tau) - y_1(\tau))]d\tau$$

Hence

$$|e^{-2t}(y_1^k(t) - y_1(t))| \leq e^{-2t}\int_0^t e^{2\tau}|e^{-2\tau}(y_1^{k-1}(\tau) - y_1(\tau))|d\tau \tag{4.6}$$

Define

$$E^k \triangleq \max_{t \in [0,T]} |e^{-2t}(y_1^k(t) - y_1(t))| \tag{4.7}$$

From (4.6) and (4.7), we have

$$\begin{aligned} |e^{-2t}(y_1^k(t) - y_1(t))| &\leq e^{-2t}\int_0^t e^{2\tau}E^{k-1}d\tau \qquad \text{for all } t \in [0,T] \\ &\leq E^{k-1}e^{-2t}\int_0^t e^{2\tau}d\tau \qquad \text{for all } t \in [0,T] \\ &\leq \frac{1}{2}E^{k-1} \qquad \text{for all } t \in [0,T] \end{aligned}$$

Therefore

$$E^k \leq \frac{1}{2}E^{k-1} \leq \frac{1}{2^k}E^0$$

Hence $\lim_{k\to\infty} E^k = 0$ which implies that the iterated solution $(y_1^k(t) ; t \in [0,T])$ always converges to the exact solution of the given system (4.2) independent of the initial guess $y_1^0(\cdot)$ and the input $u(\cdot)$. ∎

The above two examples clearly indicate that an inconsistent AP process can lead to serious convergence problems for the relaxation process of the WR algorithm and should be avoided. To give an intuitive reason why inconsistent AP processes should be avoided, consider a dynamical system which has $n$ states. Suppose that an inconsistent AP process applied to this system produces an inconsistent decomposed system having $m$ states where $m \neq n$. This means

that the natural response of the decomposed system has $m$ natural time constants whereas the natural response of the given system has $n$ natural time constants. Therefore it is not very likely that the iterated solution obtained from the relaxation process which iterates only on the decomposed system will converge to the exact solution of the given system, let alone the fact that it might not converge at all. For this reason, we shall focus only on consistent WR algorithms. Note that all the AP processes that we used in the examples of the previous chapter are consistent with the circuit equations. We shall also see later that, for large scale integrated circuits, there are simple procedures that automatically guarantee the consistency of the WR algorithms. Of course, using a consistent AP process does not necessary imply that convergence is guaranteed. Additional conditions on the consistent decomposed system to guarantee convergence of the iterated solution will be discussed in the next chapter.

Having stated that inconsistent AP processes are undesirable, the next problem that we shall address is how to obtain a consistent AP process or how to verify that an AP process is consistent or not. In general, there are two approaches to this problem, namely the *physical* approach and the *formal* approach. The physical approach is based on the physical interpretation and the physical structure of the system being considered. For example the state variables of a lumped electrical circuit are usually voltages (or charges) across capacitors and currents (or fluxes) through inductors. Hence, given the circuit topology, a circuit designer can easily identify the state variables of the circuit. This approach uses the fact that the decomposed system also has a physical interpretation (as we have demonstrated earlier in the previous chapter). Based on the physical interpretation, the state variables of the decomposed system are identified and are used to verify consistency of the AP process. In fact, to obtain an AP process, it is customary to first identify the state variables of the

given system and then choose an AP process such that these variables are also the state variables of the resulting decomposed system. On the other hand, the formal approach relies on using an algorithm to identify the state variables of the system directly from the system equations without depending on the physical interpretation. Hence it is more general than the physical approach.

## 4.2. The Formal Approach for Finding and Checking a Consistent AP Process.

In order to check the consistency criteria formally as specified by Definition 4.1, we must be able to identify directly from the system equations a set of variables that can form a state vector of the system. However, since we are dealing with a system of mixed implicit algebraic-differential equations of the most general form, i.e.,

$$F(\dot{y}, y, u) = 0,$$

we shall not attempt to determine explicitly the state equation form of the system equations. In fact, a global representation of the state equations of the system may not even exist. Therefore, in our approach, we shall identify the state vector of the system symbolically from the dependency structure of the system equations which is given in the form of an algebraic-differential dependency matrix.

**Definition 4.2** The *algebraic-differential dependency matrix* of a system of $p$ equations in $p$ unknown variables $y_1, y_2,\ldots,y_p$ is a matrix $D \in \mathbf{R}^{p\times p}$ whose $ij$-th element is given by

$D_{ij} = 0$ if the $i$-th equation does not involve $y_j$ or $\dot{y}_j$

$D_{ij} = 1$ if the $i$-th equation involves $y_j$ but not $\dot{y}_j$

$D_{ij} = 2$ if the $i$-th equation involves $\dot{y}_j$ ∎

**Definition 4.3** The *symbolic state vector* of a given system of equations whose algebraic-differential dependency matrix is $D$ is the *largest* state vector that a system of equations whose algebraic-differential dependency matrix is $D$ can have. The dimension of the symbolic state vector is called the *symbolic number of states* of the given system. ∎

For example, consider a lumped electrical circuit consisting of independent sources, capacitors and resistors. It is easy to see that the symbolic number of states of the circuit is equal to the total number of capacitors minus the total number of CE and C loops where CE loops are loops of capacitors and independent sources and C loops are loops of capacitors only. Note that in this case the symbolic state vector is also the state vector of the circuit. In most cases, it is easy to identify the symbolic state variables of any circuit by simply examining the circuit topology and the type of elements in the circuit, e.g. resistors, capacitors and inductors. However, the following example shows that, for any arbitrarily given system of equations, the symbolic state vector may be larger than the actual state vector of the system.

**Example 4.3** Consider the following system of equations

$$1.5\dot{y}_1 + 3.0\dot{y}_2 + y_1 - u_1 = 0 \tag{4.8a}$$

$$1.0\dot{y}_1 + 2.0\dot{y}_2 + y_2 - u_2 = 0 \tag{4.8b}$$

Multiplying (4.8b) by 1.5 and subtracting it from (4.8a), we obtain

$$y_1 - 1.5y_2 = u_1 - 1.5u_2 \tag{4.9}$$

which is an algebraic equation. Hence this system has only one state, i.e., either $y_1$ or $y_2$. Equation (4.9) is a conditional algebraic equation since it is induced by particular values of the coefficients that make the coefficient matrix of $\dot{y}$ singular. Notice that a slight random perturbation of the coefficients can easily destroy this conditional algebraic relation since a matrix of the same zero-nonzero

structure as the coefficient matrix of $\dot{y}$ is nonsingular for all values of the coefficients except for a set of null measure [26]. The symbolic number of states of (4.8) is thus equal to 2 and the symbolic state variables of the given system are $y_1$ and $y_2$. ■

**Assumption 4.1** (Nondegeneracy of the symbolic state vector).

Given a system of algebraic-differential equations, its state vector is a symbolic state vector. ■

The nondegeneracy assumption 4.1 makes the problem of finding a state vector of a given system tractable. This assumption is rather mild in practice. Based on this assumption, we can formulate our problem into a graph problem dealing only with the algebraic-differential dependency matrix of the given system. We first introduce a few definitions derived from the standard definitions in graph theory [24].

**Definition 4.4** The *weighted bipartite graph* associated with an algebraic-differential dependency matrix $D \in \mathbb{R}^{p \times p}$ of a given system of equations is a bipartite graph, denoted by $(S,V,B)$, with the following properties:

a) $S = V = \{1,2,\ldots,p\}$. $S$ and $V$ are the sets of nodes in the graph. $S$ corresponds to the set of indices of the system equations, i.e., the row indices of $D$, and $V$ corresponds to the set of indices of the system variables, i.e., the column indices of $D$.

b) $B = \{(s,v) \mid s \in S,\ v \in V,\ D_{sv} \neq 0\}$ represents the set of all edges joining the nodes of $S$ to the nodes of $V$.

c) The *weight* of an edge $(s,v) \in B$, denoted by $w(s,v)$, is equal to $D_{sv}$. ■

**Definition 4.5** A *matching* of a weighted bipartite graph $(S,V,B)$, denoted by $M$, is a set of edges with the property that no two edges have a node in common. If $|M| = |S| = |V|$ where $|\cdot|$ denotes the cardinality of a set, i.e., the number of edges in $M$ is equal to the number of nodes in either $S$ or $V$, $M$ is then said to be a *complete matching*. ∎

From the above definition, a matching is actually a graph-theoretic interpretation of the assignment stage of the AP process. That is, each edge of the matching represents the assignment of a system variable to a system equation. The weight of the edge indicates whether the assigned variable will be treated as a symbolic state variable or not. An example of a system of equations, its weighted bipartite graph and some complete matchings of the graph is shown in Fig. 4.1 and Fig. 4.2.

**Definition 4.6** The *weight* of a *matching* $M$ of a weighted bipartite graph $(S,V,B)$ is equal to $\sum_{(s,v)\in M} w(s,v)$, i.e., it is the sum of the weights of all edges in the matching. A complete matching $M$ is said to be a *maximum weighted* complete matching of a weighted bipartite graph if its weight is larger or equal to the weight of any other complete matching of the graph. ∎

For example, consider the systems of equations shown in Fig. 4.1 and Fig. 4.2. In both cases, it is easy to see that both $M_1$ and $M_3$ are maximum weighted complete matchings but $M_2$ is not. In Fig. 4.1, the weight of $M_1$ or $M_3$ is 6 whereas the weight of $M_1$ or $M_3$ in Fig. 4.2 is 5.

The following lemma gives a graph property of a maximum weighted complete matching which will be useful in checking whether an AP process is consistent or not.

**Lemma 4.1** Define an *alternating cycle* with respect to a matching $M$ of a given weighted bipartite graph $(S,V,B)$ as a set of edges, denoted by $L$, such that it forms a simple cycle (or loop) in the graph and that no two edges of $L \cap \overline{M}$ have a node in common, i.e., the cycle is formed by alternating edges from $M$ and $\overline{M}$ where $\overline{M}$ denotes the complement of $M$. Then $M$ is a maximum weighted complete matching if and only if, for any alternating cycle $L$ with respect to $M$,

$$\{\sum w(s,v) \mid (s,v) \in L \cap M\} \geq \{\sum w(s,v) \mid (s,v) \in L \cap \overline{M}\} \qquad (4.10)$$

i.e., the total weight of edges in the alternating cycle that belong to $M$ is larger than or equal to the total weight of edges in the cycle that do not belong to $M$. ∎

We now give the following result which states that the symbolic state vector of a given system of equations is associated with a maximum weighted complete matching of its weighted bipartite graph.

**Lemma 4.2** Given a system of $p$ algebraic-differential equations in $p$ unknown variables $y_1, y_2, \ldots, y_p$ and its weighted bipartite graph $(S,V,B)$, let $M$ be a maximum weighted complete matching of the graph. Then the set of variables $\{y_v \mid (s,v) \in M \text{ and } w(s,v) = 2\}$ is a set of the symbolic state variables of the system and $\sigma - p$ is the symbolic number of states where $\sigma$ is the weight of $M$. If the system also satisfies the nondegeneracy assumption 4.1, then the set $\{y_v \mid (s,v) \in M \text{ and } w(s,v) = 2\}$ forms a state vector of the system. ∎

For example, in Fig. 4.1, the weight of the maximum weighted complete matching $M_1$ or $M_3$ is 6, indicating that the symbolic number of states of this system is 6-3 = 3. Hence, provided that the coefficient matrix of $\dot{y}$ is nonsingular, the set of the state variables of the system is $\{y_1, y_2, y_3\}$. In Fig. 4.2, the weight of the maximum weighted complete matching $M_1$ or $M_3$ is 5, indicating that the symbolic number of states of the system is 5-3 = 2. Hence, provided that the coefficient matrix of $\dot{y}$ attains its maximum rank (which is 2), the set of state variables of the system is either $\{y_1, y_3\}$ or $\{y_1, y_2\}$.

We are now ready to describe an algorithm for finding a consistent AP process. The basic idea behind this algorithm is to maintain the symbolic state vector of the given system as a symbolic state vector of the decomposed system. Hence, the resulting AP process is consistent with the given system if both the given system and the decomposed system satisfy the nondegeneracy assumption 4.1.

**Algorithm** 4.1 (Algorithm for Finding a Consistent AP Process)

Step 1: Find a maximum weighted complete matching $M$ of the weighted bipartite graph associated with the given system of equations

Step 2: Select the assignment according to $M$ and the state variables according to Lemma 4.2.

Step 3: Perform the partitioning of the system equations into subsystems of equations. ■

**Remarks**

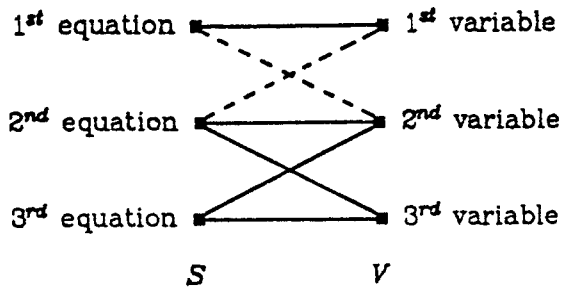
1) The problem of finding a maximum weighted complete matching is a standard problem in combinatorial optimization [24] and there are efficient algorithms for it.

2) By assigning the variables according to the matching $M$ given by Step 1, we guarantee that $M$ is also a maximum weighted complete matching of the weighted bipartite graph associated with the decomposed system independent of the choice of partitioning of Step 3. ■

To check if an AP process is consistent or not, first map the assignment of the given AP process into a complete matching of the weighted bipartite graph and apply the result of Lemma 4.1. If the matching satisfies (4.10) of the lemma, then it is a maximum weighted complete matching and, by Lemma 4.2, we can conclude that the AP process is consistent with the system.

$$\begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & a_{23} \\ 0 & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & 0 \\ b_{21} & b_{22} & b_{23} \\ 0 & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = 0$$

**(a)**




**(b)**

$$M_1 = \{(1,1), (2,2), (3,3)\}$$
$$M_2 = \{(1,2), (2,1), (3,3)\}$$
$$M_3 = \{(1,1), (2,3), (3,2)\}$$

**(c)**

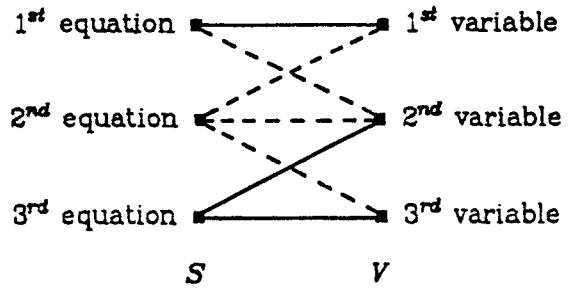
**Fig. 4.1 a)** The system equations.

**b)** The weighted bipartite graph associated with the system. Solid lines represent edges with weight=2 and broken lines represent edges with weight=1.

**c)** Some complete matchings of the graph.

$$\begin{bmatrix} a_{11} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & 0 \\ b_{21} & b_{22} & b_{23} \\ 0 & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = 0$$

**(a)**




**(b)**

$$M_1 = \{(1,1), (2,2), (3,3)\}$$
$$M_2 = \{(1,2), (2,1), (3,3)\}$$
$$M_3 = \{(1,1), (2,3), (3,2)\}$$

**(c)**

**Fig. 4.2 a)** The system equations.

**b)** The weighted bipartite graph associated with the system. Solid lines represent edges with weight=2 and broken lines represent edges with weight=1.

**c)** Some complete matchings of the graph.

# Chapter 5

# Convergence of the WR Method

A WR algorithm applied to a given dynamical system is said to *converge* if it generates a converging sequence of iterated solutions whose limit is the solution of the given system with the given initial conditions. In this chapter we give sufficient conditions on the decomposed system to guarantee convergence of the WR algorithm. The key principle behind them is the well known mathematical concept of *contraction* property of a map. To be able to give the convergence conditions of the WR algorithm in a simplified form, we shall introduce the definition of the *canonical WR algorithm*. Sufficient conditions to guarantee the existence of a canonical WR algorithm will also be given.

## 5.1. Contraction Theorems in Functional Space.

From an abstract viewpoint, the WR method can be considered as a *fixed point algorithm* [20] or a *method of successive approximations* for finding a *fixed point* [20] of a map in a functional space of waveforms. To illustrate this point, we define $Y$ as a space of waveforms within a given time interval $[0,T]$, i.e.,

$$Y = \{y(\cdot) : [0,T] \rightarrow \mathbf{R}^p\} \tag{5.1}$$

Next, we define a map $\mathbf{F}: Y \rightarrow Y$ such that $\mathbf{F}(y(\cdot))$ is the solution of the decomposed system with the given initial condition and with $y(\cdot)$ as the guess in computing the decoupling vector of each decomposed subsystem. Then the relaxation iteration of a WR algorithm can be rewritten as

$$y^k(\cdot) = \mathbf{F}(y^{k-1}(\cdot)) \tag{5.2}$$

Clearly, if $\hat{y}(\cdot)$ is the exact solution of the given system, we have

$$\hat{y}(\cdot) = \mathbf{F}(\hat{y}(\cdot)) \tag{5.3}$$

That is, the solution of the given system is a fixed point of the map **F** and the WR algorithm, as described in the form of (5.2), is called a fixed point algorithm.

The sufficient condition for convergence of a fixed point algorithm is based on a well known property of the map **F** whose fixed point is being sought. This property is called the *contraction* property [20] defined as follows.

**Definition 5.1** Let $Y$ be a complete normed space [26] (or a Banach space [26]). A map $\mathbf{F}: Y \to Y$ is *contractive* if there is a constant $\gamma \in [0,1)$ such that

$$\|\mathbf{F}(y) - \mathbf{F}(x)\| \leq \gamma \|y - x\| \qquad \text{for all} \quad x,y \in Y. \qquad \blacksquare$$

**Theorem 5.1 (Contraction Mapping Theorem)**

Let $Y$ be a complete normed space and $\mathbf{F}: Y \to Y$ be a contraction map. Then **F** has a unique fixed point $\hat{y} \in Y$ satisfying $\hat{y} = \mathbf{F}(\hat{y})$. Furthermore, for any initial guess $y^0 \in Y$, the sequence $\{y^k \in Y\}_{k=1}^{\infty}$ generated by the fixed point algorithm

$$y^k = \mathbf{F}(y^{k-1}) \qquad\qquad k = 1,2,\ldots,\infty$$

converges uniformly to $\hat{y}$ and the rate of convergence is given by

$$\|y^k - \hat{y}\| \leq \gamma^k \|y^0 - \hat{y}\| \tag{5.4}$$

where $\gamma$ is the contraction constant of **F**. $\blacksquare$

The Contraction Mapping Theorem [20] is a well known theorem in mathematics. Since it is of importance for the proof of the convergence of the WR algorithm, we shall review the proof of this theorem in the appendix. This theorem will serve as a fundamental mean for deriving sufficient conditions for the convergence of the WR algorithm in terms of the numerical conditions of the decomposed system equations as we shall see in the next section.

In practice, due to rounding or discretization errors in evaluating **F**, an approximate sequence is generated in place of the exact sequence, i.e., $y^{k+1} \neq \mathbf{F}(y^k)$. Furthermore, the map **F** itself can also be sequentially approximated. The next theorem states that if **F** is contractive, there is an adaptive scheme for controlling the errors due to these approximations which will generate a sequence of solutions that converges to the fixed point of **F**.

**Theorem 5.2** Given that **F** and $\mathbf{F}_k$ ; $k = 0,1,\ldots,\infty$ are contraction maps from a completed normed space $Y$ to $Y$ with contraction constants $\gamma$ and $\gamma_k$ ; $k = 0, 1,\ldots,\infty$ respectively, let $\hat{y} \in Y$ be the unique fixed point of **F**, i.e., $\mathbf{F}(\hat{y}) = \hat{y}$, and $\{y^k \in Y\}_{k=0}^{\infty}$ be a sequence in $Y$. Define

$$\varepsilon_k \triangleq \|\mathbf{F}_k(y^k) - y^{k+1}\| \qquad k = 0,1,\ldots,\infty \tag{5.5}$$

$$\delta_k \triangleq \|\mathbf{F}_k(\hat{y}) - \mathbf{F}(\hat{y})\| \qquad k = 0,1,\ldots,\infty \tag{5.6}$$

$$\alpha_k \triangleq \varepsilon_k + \delta_k \qquad k = 0,1,\ldots,\infty \tag{5.7}$$

and

$$\beta_{kj} \triangleq \begin{cases} \prod_{i=j+1}^{k} \gamma_i & \text{if } 0 \leq j \leq k-1 \\ 1 & \text{if } j \geq k \end{cases} \tag{5.8}$$

Then the following statements hold.

a) $$\|y^{k+1} - \hat{y}\| \leq \beta_{k0}\gamma_0\|y^0 - \hat{y}\| + \sum_{j=0}^{k} \beta_{kj}\alpha_j \tag{5.9}$$

b) $$\|y^{k+1} - \hat{y}\| \leq \frac{\gamma_k}{1-\gamma_k}\|y^{k+1} - y^k\| + \frac{\alpha_k}{1-\gamma_k} \tag{5.10}$$

c) If $\lim_{k\to\infty} \alpha_k = 0$, $\lim_{k\to\infty} \beta_{kj} = 0$ for any $j$ and $\lim_{k\to\infty} \sum_{j=0}^{k} \beta_{kj} = c < \infty$,

then $\lim_{k\to\infty} y^k = \hat{y}$.

**Corollary 5.2** If $\gamma_k \leq \tilde{\gamma} < 1$ for all $k = 0,1,\ldots,\infty$ and $\lim_{k\to\infty} \alpha_k = 0$, then

$$\lim_{k\to\infty} y^k = \hat{y}.$$

■

**Remark** Note that (5.9) and (5.10) have different uses. Whereas (5.9) is used to prove the convergence of the approximate sequence, (5.10) may serve as as a computable estimate which may be used to terminate the iteration. ∎

Careful examination of Theorem 5.2 indicates that $\varepsilon_k$ denotes the error due to the evaluation of $\mathbf{F}_k$ whereas $\delta_k$ is somewhat related to how accurate $\mathbf{F}_k$ approximates $\mathbf{F}$. This theorem is of particular importance for the WR method since practical implementation of a WR algorithm always entails some numerical errors. The most obvious error is the error due to the discretization of the time derivative in the integration method and hence can be represented by $\varepsilon_k$. Moreover, $\delta_k$ can also be used to represent the errors caused by solving a simplified decomposed system, i.e., the equations describing the decomposed system are simplified. Theorem 5.2 states that if the conceptual (or ideal) version of a WR algorithm satisfies the Contraction Mapping Theorem then convergence of its implementable version is still guaranteed if these errors are eventually driven down to zero.

## 5.2. Convergence of the Canonical WR Algorithm.

In this section, we shall derive sufficient conditions to guarantee convergence of a WR algorithm in terms of the numerical properties of the iterated equations. However, to be able to obtain interesting and useful results, it is often necessary to make certain restrictions on the object that we study. Here we require that the iterated equations of a WR algorithm can be transformed into a *canonical* form as defined below.

**Definition 5.2** A *canonical WR algorithm* is characterized by the following iterated equations.

$$\dot{x}^k = f(x^k, x^{k-1}, \dot{x}^{k-1}, z^{k-1}, u) \tag{5.11a}$$

$$z^k = g(x^k, x^{k-1}, \dot{x}^{k-1}, z^{k-1}, u) \tag{5.11b}$$

where $x \in \mathbb{R}^n$ is the vector of the state variables, $z \in \mathbb{R}^l$, $u \in \mathbb{R}^r$ and $f$, $g$ are continuous functions. Note that the time derivative of the non-state variable $z$ of the canonical WR algorithm does not appear in (5.11). ∎

It should be noted that, when simulating large scale integrated circuits, this restriction can always be met as we shall see in the next chapter. Moreover, as we shall see later, we do not have to implement a WR algorithm in its canonical form, i.e., we are not required to find $f$ and $g$ explicitly. However, the canonical form is important to simplify the derivation of sufficient conditions for the convergence of the WR algorithm.

**Theorem 5.3 (Convergence Theorem of the Canonical WR Algorithms).**

Consider a WR algorithm whose iterated equations can be transformed into the following canonical form:

$$\dot{x}^k = f(x^k, x^{k-1}, \dot{x}^{k-1}, z^{k-1}, u); \quad x^k(0) = x_0 \tag{5.12a}$$

$$z^k = g(x^k, x^{k-1}, \dot{x}^{k-1}, z^{k-1}, u) \tag{5.12b}$$

where $x^k \in \mathbb{R}^n$, $z^k \in \mathbb{R}^l$, $x_0 \in \mathbb{R}^n$ and $u \in \mathbb{R}^r$. Assume that

a) $u(\cdot) : [0,T] \to \mathbb{R}^r$ is a piecewise continuous[1] function.

b) there exist norms in $\mathbb{R}^n \times \mathbb{R}^l$ and $\mathbb{R}^n$, $\lambda_1 \geq 0$, $\lambda_2 \geq 0$ and $\gamma \in [0,1)$ such that for any $a, b, s, \tilde{a}, \tilde{b}, \tilde{s} \in \mathbb{R}^n$, $v, \tilde{v} \in \mathbb{R}^l$ and $u \in \mathbb{R}^r$

$$\left\| \begin{matrix} f(a,b,s,v,u) - f(\tilde{a},\tilde{b},\tilde{s},\tilde{v},u) \\ g(a,b,s,v,u) - g(\tilde{a},\tilde{b},\tilde{s},\tilde{v},u) \end{matrix} \right\| \leq \lambda_1 \|a - \tilde{a}\| + \lambda_2 \|b - \tilde{b}\| + \gamma \left\| \begin{matrix} s - \tilde{s} \\ v - \tilde{v} \end{matrix} \right\|$$

i.e., $(f, g)$ is globally Lipschitz continuous with respect to $x$ and globally contractive with respect to $(\dot{x}, z)$.

[1] A function $u(\cdot) : [0,T] \to \mathbb{R}^r$ is *piecewise continuous* if it is continuous everywhere except at a finite number of points and at any discontinuity point, the function has finite left- and right-hand limits.

c) both $f$ and $g$ are continuous with respect to $u$.

Then, for any initial guess $(x^0(t), z^0(t) ; t \in [0,T])$ such that both $\dot{x}^0(\cdot)$ and $z^0(\cdot)$ are piecewise continuous waveforms and that $x^0(0) = x_0$, the sequence $\{(\dot{x}^k(t), x^k(t), z^k(t) ; t \in [0,T])\}_{k=1}^{\infty}$ generated by the WR algorithm converges uniformly to $(\dot{\hat{x}}(t), \hat{x}(t), \hat{z}(t) ; t \in [0,T])$ which satisfies

$$\dot{\hat{x}} = f(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) ; \quad \hat{x}(0) = x_0 \tag{5.13a}$$

$$\hat{z} = g(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) \tag{5.13b}$$

■

**Remark:** We do not need condition (b) of Theorem 5.3 to hold for the entire spaces $R^n \times R^l$ and $R^n$, i.e., global Lipschitz and global contractive properties of $(f, g)$ are not necessary. Convergence is still guaranteed as long as the condition (b) holds for the subsets of $R^n \times R^l$ and $R^n$ that contain the sequences $\{(\dot{x}^k(t), z^k(t) ; t \in [0,T])\}_{k=0}^{\infty}$ and $\{(x^k(t) ; t \in [0,T])\}_{k=0}^{\infty}$ respectively. ■

It is possible to justify intuitively the derivation of the convergence conditions given in Theorem 5.3 if one is familiar with the Contraction Mapping Theorem and the Picard-Lindelöf Theorem on the existence and uniqueness of the solutions of ordinary differrential equations (see [25] page 18). From the Contraction Mapping Theorem, the conditions (b) and (c) guarantee that (5.12) can be written equivalently as

$$\dot{\hat{x}} = \hat{f}(\hat{x}, u) ; \quad \hat{x}(0) = x_0 \tag{5.14a}$$

$$\hat{z} = \hat{g}(\hat{x}, u) \tag{5.14b}$$

where $\hat{f}$ and $\hat{g}$ are Lipschitz continuous with respect to $\hat{x}$ and continuous with respect to $u$. Hence, by the Picard-Lindelöf Theorem, (5.14) has a unique solution $(\hat{x}, \hat{z})$ for any given initial condition and any given piecewise continuous input. Theorem 5.3 simply shows that the canonical WR algorithm is in fact a constructive proof of the existence and uniqueness of the solution.

The next theorem extends the result of the above convergence theorem to the case in which the initial conditions of the iterated decomposed system are not fixed but vary from one iteration to another. Such a case can arise when the initial conditions of the original system are not given explicitly but have to be obtained from another iterative solution of a set of algebraic equations. For example, the initial conditions of an electronic circuit may be given implicitly as the "dc" solution of the circuit equations. In such case, we may opt to use the iterated solution of the algebraic equations as the "iterated" initial conditions for solving the iterated decomposed system, allowing both iterative processes to interleave among themselves. We will see that, under the same assumptions of Theorem 5.3, the sequence of iterated solutions is guaranteed to converge to the correct solution if the sequence of the "iterated" initial conditions converges to the correct initial conditions of the original system.

**Theorem 5.4** Consider the canonical WR algorithm as given in Theorem 5.3 with the exception that the initial conditions of the iterated equations are given by

$$x^k(0) = x_0^k \quad \text{with} \quad \lim_{k\to\infty} x_0^k = x_0.$$

Then, under the same assumptions of Theorem 5.3, the sequence $\{(\dot{x}^k(t), x^k(t), z^k(t) ; t \in [0,T])\}_{k=1}^{\infty}$ generated by the canonical WR algorithm converges uniformly to the solution of (5.13). ∎

So far, the WR algorithm that we have described can be considered as being *stationary* in the sense that the iteration process is performed with the same set of equations. We now briefly describe *non-stationary* WR algorithms in which the equations describing the system at each iteration can change from one iteration to another. Obviously, nonstationary WR algorithms are meaningful only if they can be interpreted as approximations of a stationary one. The following theorem tells us how to construct a nonstationary WR algorithm from a con-

vergent stationary one without losing the convergence property.

**Theorem 5.5 (Convergence of Non-stationary WR Algorithms)**

Let $(\hat{x}(t), \hat{z}(t) : t \in [0,T])$ be the solution of (5.13) in which $f$, $g$ and $u$ satisfy all the assumptions of Theorem 5.3. Let $\{(x^k(t), z^k(t) : t \in [0,T])\}_{k=1}^{\infty}$ be the sequence of iterated solutions satisfying the following non-stationary WR algorithm:

$$\dot{x}^k = f^k(x^k, x^{k-1}, \dot{x}^{k-1}, z^{k-1}, u) \; ; \quad x^k(0) = x_0 \tag{5.15a}$$

$$z^k = g^k(x^k, x^{k-1}, \dot{x}^{k-1}, z^{k-1}, u) \tag{5.15b}$$

where $x^k \in R^n$, $z^k \in R^l$, $x_0 \in R^n$, $u \in R^r$, $f^k : R^n \times R^n \times R^n \times R^l \times R^r \to R^n$ and $g^k : R^n \times R^n \times R^n \times R^l \times R^r \to R^l$. Assume that

a) for each $k$, there exist constants $\lambda_{1k} \geq 0$, $\lambda_{2k} \geq 0$ and $\gamma_k \in [0,1)$ such that

$$\left\| \begin{matrix} f^k(a, b, s, v, u) - f^k(\tilde{a}, \tilde{b}, \tilde{s}, \tilde{v}, u) \\ g^k(a, b, s, v, u) - g^k(\tilde{a}, \tilde{b}, \tilde{s}, \tilde{v}, u) \end{matrix} \right\| \leq \lambda_{1k}\|a - \tilde{a}\| + \lambda_{2k}\|b - \tilde{b}\| + \gamma_k \left\| \begin{matrix} s - \tilde{s} \\ v - \tilde{v} \end{matrix} \right\|$$

for all $a, b, s, \tilde{a}, \tilde{b}, \tilde{s} \in Rn$, $v, \tilde{v} \in R^l$ and $u \in R^r$ where all norms are the same as in Theorem 5.3.

b) $\gamma_M \triangleq \sup\{\gamma_k \mid k = 1,2,\ldots,\infty\} < 1$, $\lambda_{1M} \triangleq \sup\{\lambda_{1k} \mid k = 1,2,\ldots,\infty\} < \infty$ and $\lambda_{2M} \triangleq \sup\{\lambda_{2k} \mid k = 1,2,\ldots,\infty\} < \infty$

c) for any $t \in [0,T]$, $\lim_{k\to\infty} \xi_k(t) = 0$ where

$$\xi_k \triangleq \left\| \begin{matrix} f(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) - f^k(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) \\ g(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) - g^k(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) \end{matrix} \right\|$$

Then, for any initial guess $(x^0(t), z^0(t) : t \in [0,T])$ such that both $\dot{x}^0(\cdot)$ and $z^0(\cdot)$ are piecewise continuous waveforms and that $x^0(0) = x_0$, the sequence $\{(\dot{x}^k(t), x^k(t), z^k(t) : t \in [0,T])\}_{k=1}^{\infty}$ generated by the non-stationary WR algorithm converges uniformly to $(\dot{\hat{x}}(t), \hat{x}(t), \hat{z}(t) : t \in [0,T])$. ∎

**Corollary 5.5** The result of Theorem 5.5 still holds when the initial condition of (5.15) is given by $x^k(0) = x_0^k$, provided that $\lim_{k\to\infty} x_0^k = x_0$. ∎

We conclude this section with the remark that we have presented the WR method in this and the previous chapter in a mathematical framework which is quite general. However, since we made no assumption about the structure of the system equations, we had to make certain theoretical restrictions ( such as the consistency requirement and the existence of a canonical representation ) in order to be able to derive the convergence conditions of the WR method. This makes it difficult to interpret them in terms of physical properties. Moreover, there are many different ways to formulate the circuit differential equations such as Nodal Analysis formulation [23], Modified Nodal Analysis formulation [21] and Tableau formulation [22]. Each formulation differs from the other by the choice of the circuit variables. It is quite possible that the WR algorithms derived from different formulations of the circuit equations have different convergent properties. In the next chapter, we shall specialize the method to a particular formulation and a particular class of circuits. There we will see that the theoretical restrictions given in this chapter are automatically satisfied.

## 5.3. Existence of the Canonical WR Algorithm.

The convergence theorems presented in the previous section all require the existence of a canonical form of the WR algorithm eventhough the algorithm is actually implemented in its original form. In this section, we shall show that the consistency of a WR algorithm plays an important role in determining the existence of the canonical form of the algorithm. We first give a few examples to illustrate the basic assumptions and basic steps in transforming a WR algorithm into its canonical form as defined in Definition 5.2.

**Example 5.1** Consider the WR algorithm used in Example 3.1, as decribed by (3.8). Notice that this algorithm is consistent with the original circuit equations (3.7). To transform it into a canonical WR algorithm, first rewrite (3.8a) as follows:

$$\dot{v}_1^k = \frac{J}{C_1 + C_3} - \frac{G_1}{C_1 + C_3} v_1^k + \frac{C_3}{C_1 + C_3} \dot{v}_2^{k-1} \quad ; \quad v_1^k(0) = V_1 \qquad (5.16a)$$

Then, substitute it into (3.8b) to obtain

$$\dot{v}_2^k = -\frac{G_2}{C_2 + C_3} v_2^k + \frac{C_3}{(C_2 + C_3)(C_1 + C_3)} [J - G_1 v_1^k + C_3 \dot{v}_2^{k-1}] \; ; \quad v_2^k(0) = V_2 \qquad (5.16b)$$

The canonical form of the algorithm is thus given by (5.16). ■

**Example 5.2** Consider the following system of equations:

$$\dot{y}_1 = f_1(y_1, \dot{y}_2, u) \qquad (5.17a)$$

$$y_2 = f_2(y_1, y_3) \qquad (5.17b)$$

$$y_3 = f_3(y_1, y_2) \qquad (5.17c)$$

Let $y_1$, $y_2$ and $y_3$ be assigned to (5.17a), (5.17b) and (5.17c) respectively and let (5.17) be partitioned into 3 subsystems consisting of {(5.17a)}, {(5.17b)} and {(5.17c)}. Then the $k$-th iteration of the resulting GJ-WR algorithm is described by

$$\dot{y}_1^k = f_1(y_1^k, \dot{y}_2^{k-1}, u) \; ; \qquad y_1^k(0) = y_1(0) \qquad (5.18a)$$

$$y_2^k = f_2(y_1^{k-1}, y_3^{k-1}) \qquad (5.18b)$$

$$y_3^k = f_3(y_1^{k-1}, y_2^{k-1}) \qquad (5.18c)$$

It is easy to see that the given system of equations (5.17) has only one state despite the fact that there are two variables with time derivative. Hence, this WR algorithm is consistent with the given system equations. However, it is not in

canonical form because the non-state variable $y_2$ appears in (5.17a) with a time derivative. To transform (5.18) into a canonical form, we have to differentiate (5.18b) and (5.18c). By differentiating the two equations, (5.18) can be written as

$$\dot{y}_1^k = f_1(y_1^k, \dot{y}_2^{k-1}, u) \quad ; \qquad y_1^k(0) = y_1(0) \tag{5.19a}$$

$$\dot{y}_2^k = \frac{\partial f_2}{\partial y_1}(y_1^{k-1}, y_3^{k-1})\dot{y}_1^{k-1} + \frac{\partial f_2}{\partial y_3}(y_1^{k-1}, y_3^{k-1})\dot{y}_3^{k-1} \tag{5.19b}$$

$$\dot{y}_3^k = \frac{\partial f_3}{\partial y_1}(y_1^{k-1}, y_2^{k-1})\dot{y}_1^{k-1} + \frac{\partial f_3}{\partial y_2}(y_1^{k-1}, y_2^{k-1})\dot{y}_2^{k-1} \tag{5.19c}$$

which is a canonical WR algorithm. Notice that the system of equations (5.19) of the canonical WR algorithm has 3 state variables as opposed to only one state variable in the original system. This is because additional states variables $y_2$ and $y_3$ are created by the differentiation. The initial values of $y_2^k$ and $y_3^k$ for (5.19) are given by the iterated equations (5.18b) and (5.18c) of the original WR algorithm, i.e.,

$$y_2^k(0) = f_2(y_1(0), y_3^{k-1}(0)) \tag{5.20b}$$

$$y_3^k(0) = f_3(y_1(0), y_2^{k-1}(0)) \tag{5.20c}$$

So we have a canonical WR algorithm in which the initial values of some of the state variables will change from one iteration to another unless the initial guesses $y_2^0(0)$ and $y_3^0(0)$ are equal to their exact initial values. That is, if $y_2^0(0) = y_2(0)$ and $y_3^0(0) = y_3(0)$, then

$$y_2^k(0) = y_2(0) \qquad \text{and} \qquad y_2^k(0) = y_3(0) \quad \text{for any } k \tag{5.21}$$

Now suppose that (5.19) satisfies the assumptions of the convergence theorem 5.3, which is equivalent to saying that

$$\begin{bmatrix} 0 & \frac{\partial f_1}{\partial \dot{y}_2}(y_1, \dot{y}_2, u) & 0 \\ \frac{\partial f_2}{\partial y_1}(y_1, y_3) & 0 & \frac{\partial f_2}{\partial y_3}(y_1, y_3) \\ \frac{\partial f_3}{\partial y_1}(y_1, y_2) & \frac{\partial f_3}{\partial y_2}(y_1, y_2) & 0 \end{bmatrix}$$ is uniformly contractive.

This automatically implies that its principal minor

$$\begin{bmatrix} 0 & \frac{\partial f_2}{\partial y_3}(y_1, y_3) \\ \frac{\partial f_3}{\partial y_2}(y_1, y_2) & 0 \end{bmatrix}$$ is also uniformly contractive,

which is sufficient to guarantee that the sequences of $y_2^k(0)$ and $y_3^k(0)$ as generated by (5.20) will converge to $y_2(0)$ and $y_3(0)$ respectively. Therefore, by Theorem 5.4, the conditions for convergence of the canonical WR algorithm do not depend on whether the initial values of $y_2^k$ and $y_3^k$ are given by (5.20) or (5.21). ∎

**Example 5.3** Consider the following system of equations

$$\dot{y}_1 = f_1(y_1, \dot{y}_4, u) \tag{5.22a}$$

$$\dot{y}_2 = f_2(y_2, \dot{y}_3) \tag{5.22b}$$

$$\dot{y}_3 = f_3(y_3, \dot{y}_2) \tag{5.22c}$$

$$y_4 = f_4(\dot{y}_2, y_3) \tag{5.22d}$$

Let $y_1$, $y_2$, $y_3$ and $y_4$ be assigned to (5.22a), (5.22b), (5.22c) and (5.22d) respectively and let (5.22) be partitioned into 4 subsystems consisting of {(5.22a)}, {(5.22b)}, {(5.22c)} and {(5.22d)}. The $k$-th iteration of the resulting GJ-WR algorithm is then given by

$$\dot{y}_1^k = f_1(y_1^k, \dot{y}_4^{k-1}, u) \; ; \qquad y_1^k(0) = y_1(0) \tag{5.23a}$$

$$\dot{y}_2^k = f_2(y_2^k, \dot{y}_3^{k-1}) \; ; \qquad y_2^k(0) = y_2(0) \tag{5.23b}$$

$$\dot{y}_3^k = f_3(y_3^k, \dot{y}_2^{k-1}) \quad ; \quad y_3^k(0) = y_3(0) \tag{5.23c}$$

$$y_4^k = f_4(\dot{y}_2^{k-1}, y_3^{k-1}) \tag{5.23d}$$

Note that this WR algorithm is consistent with the given system equations (5.22) but is not in canonical form because the non-state variable $y_4$ appears in (5.22a) with a time derivative. To transform (5.23) into a canonical form, we differentiate (5.23b), (5.23c) and (5.23d) to get

$$\ddot{y}_2^k = \frac{\partial f_2}{\partial y_2}(y_2^k, \dot{y}_3^{k-1})\dot{y}_2^k + \frac{\partial f_2}{\partial \dot{y}_3}(y_2^k, \dot{y}_3^{k-1})\ddot{y}_3^{k-1} \tag{5.23e}$$

$$\ddot{y}_3^k = \frac{\partial f_3}{\partial y_3}(y_3^k, \dot{y}_2^{k-1})\dot{y}_3^k + \frac{\partial f_3}{\partial \dot{y}_2}(y_3^k, \dot{y}_2^{k-1})\ddot{y}_2^{k-1} \tag{5.23f}$$

$$\dot{y}_4^k = \frac{\partial f_4}{\partial \dot{y}_2}(\dot{y}_2^{k-1}, y_3^{k-1})\ddot{y}_2^{k-1} + \frac{\partial f_4}{\partial y_3}(\dot{y}_2^{k-1}, y_3^{k-1})\dot{y}_3^{k-1} \tag{5.23g}$$

Let

$$y_5 = \dot{y}_2 \qquad \text{and} \qquad y_6 = \dot{y}_3$$

Then (5.23) can be tranformed into the following canonical form

$$\dot{y}_1^k = f_1(y_1^k, \dot{y}_4^{k-1}, u) \; ; \quad y_1^k(0) = y_1(0) \tag{5.24a}$$

$$\dot{y}_2^k = f_2(y_2^k, y_6^{k-1}) \quad ; \quad y_2^k(0) = y_2(0) \tag{5.24b}$$

$$\dot{y}_3^k = f_3(y_3^k, y_5^{k-1}) \quad ; \quad y_3^k(0) = y_3(0) \tag{5.24c}$$

$$\dot{y}_4^k = \frac{\partial f_4}{\partial \dot{y}_2}(y_5^{k-1}, y_3^{k-1})\dot{y}_5^{k-1} + \frac{\partial f_4}{\partial y_3}(y_5^{k-1}, y_3^{k-1})y_6^{k-1} \tag{5.24d}$$

$$\dot{y}_5^k = \frac{\partial f_2}{\partial y_2}(y_2^k, y_6^{k-1})y_5^k + \frac{\partial f_2}{\partial \dot{y}_3}(y_2^k, y_6^{k-1})\dot{y}_6^{k-1} \tag{5.24e}$$

$$\dot{y}_6^k = \frac{\partial f_3}{\partial y_3}(y_3^k, y_5^{k-1})y_6^k + \frac{\partial f_3}{\partial \dot{y}_2}(y_3^k, y_5^{k-1})\dot{y}_5^{k-1} \tag{5.24f}$$

Just like the previous example, the system equations of the canonical WR algorithm has more state variables than that of the original WR algorithm. The initial values of $y_4^k$ $y_5^k$ and $y_6^k$ for (5.24) are given by the iterated equations (5.23d), (5.23b) and (5.23c) of the original WR algorithm respectively, i.e.,

$$y_4^k(0) = f_4(y_5^{k-1}(0), \dot{y}_3(0)) \tag{5.25a}$$

$$y_5^k(0) = f_2(y_2(0), y_6^{k-1}(0)) \tag{5.25b}$$

$$y_6^k(0) = f_3(y_3(0), y_5^{k-1}(0)) \tag{5.25c}$$

Once again, by the arguments similar to those of the previous example, we can conclude that if (5.24) satisfies all the assumptions of Theorem 5.3 then the canonical WR algorithm will converge to the correct solution even if the initial values of some of its state variables vary from one iteration to another. ∎

**Example 5.4** Consider the WR algorithm described in example 4.1 of the previous chapter, i.e.,

$$y_2^k = u - \dot{y}_1^{k-1} \tag{5.26a}$$

$$y_1^k = y_2^k \tag{5.26b}$$

Notice that this algorithm is not a consistent WR algorithm. It is also not in canonical form because the non-state variable $y_1$ appears in (5.26a) with a time derivative. However, if we differentiate (5.26b) we will create a time derivative of $y_2$ which is also a non-state variable and the process of differentiating will loop indefinitely. Hence the canonical form of the WR algorithm does not exist. ∎

From these examples, we see that consistency is essential in ensuring the existence of a canonical form of a WR algorithm. We then formalize this result in the following lemma.

**Lemma 5.1** Consider a WR algorithm with the following assumptions.

a) For each decomposed subsystem described by (3.4), there exist smooth[2] functions $\tilde{f}_i$ and $\tilde{g}_i$ and a nonsingular matrix $\begin{bmatrix} E_i \\ D_i \end{bmatrix} \in \mathbf{R}^{p_i \times p_i}$ such that (3.4) can be written as

$$\dot{x}_i = \tilde{f}_i(x_i, \tilde{u}_i, u); \qquad x_i(0) = E_i(y_i(0)) \tag{5.27a}$$

$$z_i = \tilde{g}_i(x_i, \tilde{u}_i, u) \tag{5.27b}$$

$$\begin{bmatrix} x_i \\ z_i \end{bmatrix} = \begin{bmatrix} E_i \\ D_i \end{bmatrix} y_i \tag{5.27c}$$

where $x_i \in \mathbf{R}^{n_i}$, $z_i \in \mathbf{R}^{p_i - n_i}$, i.e., each decomposed subsystem has a state-equation representation.

b) Both the given system and its associated decomposed system satisfy the nondegeneracy assumption 4.1.

c) The WR algorithm is consistent, i.e., its AP process corresponds to a maximum weighted complete matching (defined in Definition 4.6) of the weighted bipartite graph associated with the given system.

Then there exists a transformation which transforms the WR algorithm into a canonical form as defined in Definition 5.2. ∎

[2] By smooth we mean that the functions are $r$ times continuously differentiable, where $r$ is as large as required by the transformation indentified by the constructional proof of the lemma.

# Chapter 6

# WR Algorithms for Simulating Large Scale Integrated Circuits

In this chapter we shall apply the WR Method to analyse an important class of dynamical systems: *MOS integrated circuits*. In fact, this was the original motivation behind the development of the WR method. A typical large scale digital circuit is usually an interconnection of several basic subcircuits called "gates". Hence decomposition techniques can be applied to the analysis of this class of circuits in the most natural way. We will propose two WR algorithms for analyzing MOS digital integrated circuits and show that, under very mild assumptions usually satisfied by practical circuits, the proposed algorithms converge. Although both GS and GJ relaxations can be used in these algorithms, the GS relaxation is preferred since it requires only one copy of the iterated solution as opposed to two copies required by the GJ relaxation. Also its speed of convergence is faster, especially for MOS circuits where unidirectional models are used MOS devices (for example see [5,6]), provided that the equations are properly ordered (see [30] for a discussion of this aspect). For the sake of simplicity, both algorithms use the simplest guessing scheme and an assignment-partitioning (AP) process in which each partitioned subsystem is a single equation. The generalization of both algorithms to allow more than one equation per subsystem is straightforward and will not be discussed.

Two basic assumptions are made to derive our algorithms:

(i) Each element in the circuit and its interconnections can be modelled by lumped (linear or nonlinear) voltage controlled capacitors, conductors and current sources.

(ii) Every (internal or external) node in the circuit has a (linear or nonlinear) capacitor, called a *grounded capacitor*, to either ground or dc supply voltage rails.

Note that, for MOS large scale integrated circuits, these assumptions are usually satisfied.

## 6.1. Nodal Circuit Equations and the WR Algorithm.

For the first WR algorithm, we use the node voltages as the circuit variables. Let the circuit to be simulated has $n$ unknown node voltages. Using Nodal Analysis formulation [23], the circuit equations can be written as follows.

$$C(v, u)\dot{v} + q(v, u) = 0 ; \qquad v(0) = V \tag{6.1}$$

where $v \in \mathbf{R}^n$ is the vector of all unknown node voltages, $V \in \mathbf{R}^n$ is the given initial values of $v$, $u \in \mathbf{R}^r$ is the vector of all inputs and their first order time derivatives, $q : \mathbf{R}^n \times \mathbf{R}^r \to \mathbf{R}^n$ is a continuous function each component of which represents the net sum of currents charging the capacitor at each node due to the conductors and the controlled current sources, $C : \mathbf{R}^n \times \mathbf{R}^r \to \mathbf{R}^{n \times n}$ is a symmetric diagonally dominant matrix-value function in which $-C_{ij}(v, u) ; i \neq j$ is the total floating capacitance between nodes $i$ and $j$, and $C_{ii}(v, u)$ is the sum of the capacitances of all capacitors connected to node $i$.

**Algorithm 6.1** *(WR algorithm for solving (6.1) from $t = 0$ to $t = T$.)*

Comment: a superscript denotes the WR iteration count and a subscript denotes the component index of a vector.

Step 1: Set $k = 1$ and $v^0(t) = V$ for all $t \in [0, T]$.

Step 2: For each $i = 1,2,\ldots,n$, solve for $\{v_i^k(t) ; t \in [0, T]\}$ from

$$\sum_{j=1}^{i} C_{ij}(v_1^k, \ldots, v_i^k, v_{i+1}^{k-1}, \ldots, v_n^{k-1}, u)\dot{v}_j^k \quad +$$

$$\sum_{j=i+1}^{n} C_{ij}(v_1^k, \ldots, v_i^k, v_{i+1}^{k-1}, \ldots, v_n^{k-1}, u)\dot{v}_j^{k-1} \quad +$$

$$q_i(v_1^k, \ldots, v_i^k, v_{i+1}^{k-1}, \ldots, v_n^{k-1}, u) \quad = \quad 0 \qquad (6.2)$$

with the initial condition $v_i^k(0) = V_i$.

Step 3: Set $k = k+1$ and go to step 2. ∎

**Remarks:**

1) Equation (6.2) is actually a single differential equation in one unknown variable $v_i^k$. The variables $v_{i+1}^{k-1}, \ldots, v_n^{k-1}$ are known from the previous iteration and $v_1^k, \ldots, v_{i-1}^k$ have already been computed.

2) In most practical circuits, the circuit equations (6.1) are usually very sparse, i.e., only a few variables are actually involved in each equation. This fact can be exploited in the implementation of the algorithm on a computer.

3) With regard to the theoretical concepts presented in the previous three chapters, we can say that the AP process used in Algorithm 6.1 assigns $v_i$ to the $i$-th equation of (6.1) and partitions (6.1) into $n$ subsystems each of which has only one equation. Since it is clear that $\{v_i,\ i = 1,2,\ldots,n\}$ form a state vector of both (6.1) and (6.2), therefore the AP process is automatically consistent with the given circuit. ∎

**Example 6.1**

Consider the circuit shown in Fig. 6.1. For simplicity we assume that all capacitors are linear. Using the node voltages $v_1$, $v_2$ and $v_3$ as variables, the circuit equations are:

$$(c_1 + c_2 + c_3)\dot{v}_1 - i_1(v_1) + i_2(v_1, u_1) + i_3(v_1, u_2, v_2) - c_1\dot{u}_1 - c_3\dot{u}_2 = 0 \quad (6.3a)$$

$$(c_4 + c_5 + c_6)\dot{v}_2 - c_6\dot{v}_3 - i_3(v_1, u_2, v_2) - c_4\dot{u}_2 = 0 \quad (6.3b)$$

$$(c_6 + c_7)\dot{v}_3 - c_6\dot{v}_2 - i_4(v_3) + i_5(v_3, v_2) = 0 \quad (6.3c)$$

Applying Algorithm 6.1, the $k$-th WR iteration corresponds to solving the following equations.

$$(c_1 + c_2 + c_3)\dot{v}_1^k - i_1(v_1^k) + i_2(v_1^k, u_1) + i_3(v_1^k, u_2, v_2^{k-1}) - c_1\dot{u}_1 - c_3\dot{u}_2 = 0 \quad (6.4a)$$

$$(c_4 + c_5 + c_6)\dot{v}_2^k - c_6\dot{v}_3^{k-1} - i_3(v_1^k, u_2, v_2^k) - c_4\dot{u}_2 = 0 \quad (6.4b)$$

$$(c_6 + c_7)\dot{v}_3^k - c_6\dot{v}_2^k - i_4(v_3^k) + i_5(v_3^k, v_2^k) = 0 \quad (6.4c)$$

The circuit interpretation of the iterated equations (6.4) is shown in Fig. 6.2.

If we consider that the original circuit in Fig. 6.1. consists of 3 subcircuits $s_1$, $s_2$ and $s_3$, then the decomposed subcircuits $\tilde{s}_1$, $\tilde{s}_2$ and $\tilde{s}_3$ (shown in Fig. 6.2) are actually $s_1$, $s_2$ and $s_3$ together with additional components to approximate their loadings. Hence we can describe the WR algorithm for simulating this circuit in circuit terms as follows.

Step 1: Set k = 0 and make an initial guess of $v_2^0(t)$, $v_3^0(t)$ ; $t \in [0, T]$.

Step 2: Repeat

Set $k = k+1$.

Analyse $s_1$ for its output waveform $v_1^k(\cdot)$ by approximating the loading effect due to $s_2$.

Analyse $s_2$ for its output waveform $v_2^k(\cdot)$ by using $v_1^k(\cdot)$ as its input and approximating the loading effect due to $s_3$.

Analyse $s_3$ for its output waveform $v_3^k(\cdot)$ by using $v_2^k(\cdot)$ as its input.

Until the difference between $\{(v_1^k(t), v_2^k(t), v_3^k(t) ; t \in [0, T]\}$ and $\{v_1^{k-1}(t), v_2^{k-1}(t), v_3^{k-1}(t) ; t \in [0, T]\}$ is sufficiently small. ■

## 6.2. Modified Nodal Equations and the WR Algorithm.

The second WR algorithm that we are about to describe is intended for MOS circuits containing pass transistors (or transmission gates) such as the circuit in Fig. 6.1. Here, we use the unknown node voltages and the drain currents of pass transistors as the circuit variables. Let the circuit to be simulated has $n$ unknown node voltages and $l$ pass transistors. Using the Modified Nodal Analysis formulation [21], the circuit equations can be written as follows:

$$C(v, u)\dot{v} + \tilde{q}(z, v, u) = 0 \; ; \qquad v(0) = V \tag{6.5a}$$

$$z - g(v, u) = 0 \tag{6.5b}$$

where $C$, $v$, $u$ and $V$ are as defined in (6.1), $z \in \mathbf{R}^l$ is the vector of the drain currents of the pass transistors, $g : \mathbf{R}^n \times \mathbf{R}^r \to \mathbf{R}^l$ is a continuous function (each component of which describes the drain current of each pass transistor in terms of its terminal node voltages) and $\tilde{q} : \mathbf{R}^l \times \mathbf{R}^n \times \mathbf{R}^r \to \mathbf{R}^n$ is a continuous function (each component of which represents the net current charging the capacitor at each node due to the pass transistors, the other conductive elements and the controlled current sources).

**Algorithm 6.2.** *(WR algorithm for solving (6.5) from $t=0$ to $t=T$)*

Comment: a superscript denotes the WR iteration count and a subscript denotes the component index of a vector.

Step 1: Set $k = 1$, $z^0(t) = 0$ and $v^0(t) = V$ for all $t \in [0, T]$.

Step 2: a) For each $i = 1,2,\ldots,n$, solve for $(v_i^k(t) \; ; \; t \in [0, T])$ from

$$\sum_{j=1}^{i} C_{ij}(v_1^k, \ldots, v_i^k, v_{i+1}^{k-1}, \ldots, v_n^{k-1}, u)\dot{v}_j^k \; +$$

$$\sum_{j=i+1}^{n} C_{ij}(v_1^k, \ldots, v_i^k, v_{i+1}^{k-1}, \ldots, v_n^{k-1}, u)\dot{v}_j^{k-1} \; +$$

$$\tilde{q}_i(z^{k-1}, v_1^k, \ldots, v_i^k, v_{i+1}^{k-1}, \ldots, v_n^{k-1}, u) = 0 \tag{6.6a}$$

with the initial condition $v_i^k(0) = V_i$.

b) Compute $z^k(t)$ ; $t \in [0,T]$ from

$$z^k = g(v^k, u) \tag{6.6b}$$

Step 3: Set $k = k+1$ and go to step 2. ■

**Remarks:**

1) Just like (6.1), Equation (6.5) is also very sparse.

2) Equations (6.6a) and (6.6b) can actually be solved together as will be demonstrated in the following example.

3) By the same arguments as those for Algorithm 6.1, we can conclude that Algorithm 6.2 is also consistent with (6.5). ■

**Example 6.2**

Once again we consider the circuit of Fig. 6.1. This time we formulate the circuit equations as follows:

$$(c_1 + c_2 + c_3)\dot{v}_1 - i_1(v_1) + i_2(v_1, u_1) + z - c_1\dot{u}_1 - c_3\dot{u}_2 = 0 \tag{6.7a}$$

$$z - i_3(v_1, u_2, v_2) = 0 \tag{6.7b}$$

$$(c_4 + c_5 + c_6)\dot{v}_2 - c_6\dot{v}_3 - i_3(v_1, u_2, v_2) - c_4\dot{u}_2 = 0 \tag{6.7c}$$

$$(c_6 + c_7)\dot{v}_3 - c_6\dot{v}_2 - i_4(v_3) + i_5(v_3, v_2) = 0 \tag{6.7d}$$

Applying Algorithm 6.2, the $k$-th WR iteration corresponds to solving the following equations.

$$(c_1 + c_2 + c_3)\dot{v}_1^k - i_1(v_1^k) + i_2(v_1^k, u_1) + z^{k-1} - c_1\dot{u}_1 - c_3\dot{u}_2 = 0 \tag{6.8a}$$

$$z^k - i_3(v_1^k, u_2, v_2^k) = 0 \tag{6.8b}$$

$$(c_4 + c_5 + c_6)\dot{v}_2^k - c_6\dot{v}_3^{k-1} - i_3(v_1^k, u_2, v_2^k) - c_4\dot{u}_2 = 0 \tag{6.8c}$$

$$(c_6 + c_7)\dot{v}_3^k - c_6\dot{v}_2^k - i_4(v_3^k) + i_5(v_3^k, v_2^k) = 0 \tag{6.8d}$$

The circuit interpretation of the equation (6.8) is shown in Fig. 6.3. Note the difference between the decompositions induced by Algorithm 6.1 ( Fig. 6.2 ) and

Algorithm 6.2 ( Fig. 6.3 ). From Fig. 6.3 we see that (6.8b) and (6.8c) can be solved together since they belong to the same subcircuit. ∎

## 6.3. Guaranteed Convergence of WR Algorithms for MOS Circuits

In this section, we interpret the sufficient conditions for convergence of WR algorithms, as given by Theorem 5.3, in terms of the properties of the elements of the circuit and show that they are very mild in practice.

**Theorem 6.1** Assume that

a) The charge-voltage characteristic of each capacitor, or the volt-ampere characteristic of each conductor, or the drain current characteristic of each MOS device is Lipschitz continuous with respect to its controlling variables,

b) $C_{min} > 0$ and $C_{max} < \infty$ where

$C_{min} \in \mathbf{R}$ is the minimum value of all grounded capacitances at any permissible values of voltages, and

$C_{max} \in \mathbf{R}$ is the maximum value of all floating capacitances between any two nodes at any permissible values of voltages,

c) The current through any controlled conductor ( e.g. the drain current of an MOS device ) is uniformly bounded throughout the relaxation process.

Then, for any MOS circuit with any given set of initial conditions, and any given piecewise continuous input $u(\cdot)$, either Algorithm 6.1 or 6.2 generates a converging sequence of iterated solutions whose limit satisfies the circuit equations and the given initial conditions. ∎

Note that the first assumption implies that for any capacitor, conductor or MOS device, its incremetal (or small signal) characteristic, i.e., capacitance, conductance or transconductance, at any permissible dc operating point must be uniformly bounded. The second assumption states that the value of any grounded capacitor must be bounded away from zero and the value of any floating capacitor must not be arbitrarily large. The third assumption implies that during the relaxation iteration, the current through any conductor or MOS device does not grow arbitrarily large. These three assumptions are very mild in practice and hence either Algorithm 6.1 or Algorithm 6.2 is guaranteed to converge for any MOS integrated circuit of practical interest. The rate of convergence is linear, a typical property of any relaxation method.

It should be pointed out that the convergence of WR algorithms can be established by Theorem 6.1 for integrated circuits implemented by other type of devices (such as bipolar transistors) as long as the circuit equations can be written in the form of the equation (6.1) or (6.5) and the assumptions of Theorem 6.1 on the branch equations are satisfied. Note also that the strong assumption of Theorem 5.3 regarding the global contractivity of $f$ in (5.12) with respect to $\dot{x}$ is automatically satisfied because $C(\cdot, \cdot)$ is strictly diagonally dominant. Moreover, the contractivity of $g$ in (5.12) with respect to $z$ is irrelevant for Algorithm 4.1 since its canonical form does not involve algebraic equations. For Algorithm 6.2, it is also irrelevant since $g$ does not depend on $z$.

In both algorithms, the initial guesses are chosen, for convenience, to be constant waveforms. From Theorem 5.3 we know that other choices of initial guesses will not destroy the guaranteed convergence of both algorithms if they are piecewise continuous waveforms. Hence, for MOS digital integrated circuits, a logic simulation could be used to generate the initial guesses for these two algorithms. It is also possible to show that, under the same assumptions of Theorem 5.3, the corresponding GJ relaxation versions of Algorithm 6.1 and 6.2

are guaranteed to converge. Moreover, a relaxation parameter $\omega$ can be introduced into these GJ-WR or GS-WR algorithms (as described in section 3.3) without destroying their guaranteed convergence, provided that $\omega \in (0, 2)$.

As a first example, the ring oscillator shown in Fig. 6.4 is used to illustrate the convergence of Algorithm 6.1. The circuit interpretation of the relaxation process is shown in Fig. 6.5. The resulting waveforms at different iterations of the algorithm are shown in Fig. 6.6a through 6.6d. Note that since the oscillator is highly non-unidirectional due to the feedback from $v_3$ to the input of the NOR gate, the convergence of the iterated solution is achieved with the number of iterations being proportional to the number of oscillating cycles of interest.

The next example, shown in Fig. 6.7a, illustrate the convergence property of a WR algorithm when being applied to a bipolar analog integrated circuit. Once again, we purposedly choose as our example the circuit in which there is a feedback from the output to the input of the circuit. This is to show that the WR method is always guaranteed to converge regardless of whether the circuit possesses any kind of feedback. In this example, the initial guess of the waveforms $v_o^0(\cdot)$ and $v_{ob}^0(\cdot)$ are

$$v_o^0(t) = V_{o_{DC}} \quad \text{and} \quad v_{ob}^0(t) = V_{ob_{DC}} \quad \text{for all } t \geq 0$$

where $V_{o_{DC}}$ and $V_{ob_{DC}}$ are the "dc" solutions of the circuit. The resulting waveforms at different iterations of the decomposed circuit (Fig. 6.7b) are shown in Fig. 6.8. From this figure, we observe that, although the iterated solution is guaranteed to converge to the exact solution, the convergence to the steady state portion of the exact solution is quite slow whereas the convergence to the transient portion of the exact solution is achieved in two iterations. This is due to the fact that the effect of capacitors in the circuit becomes negligible when the circuit almost reaches its steady state condition. Since the decomposed circuit in this case is an open loop operational amplifier, its steady state

value is highly sensitive to the input voltages. Therefore, the convergence of the iterated solution of the decomposed circuit to the exact solution at steady state condition is very slow. Fortunately, this is not the case for digital circuits because digital circuits are usually in saturation at steady state conditions. Hence their steady state values are not sensitive to small changes in the input voltages. Furthermore, as we have pointed out earlier, a logic simulation [29] can be used to provide a good initial guess for the WR iteration, especially for digital circuits with logic feedback between subcircuits to be decomposed. For these reasons, we choose to implement the WR method for simulating digital circuits as a first practical application of the WR method in circuit simulation.

## 6.4. WR Algorithm with Adaptive MOS Models.

It is well known [3] that the computational cost of evaluating the MOS model equations can be quite expensive when the model equations contain complicated mathematical expressions. For this reason, many simulators trade off the speed of simulation with the accuracy of simulation by providing the user with various models of different accuracy. The simplest device model is probably the so called *table look-up* model [13] which is simply a piecewise linear function on uniformly subdivided intervals. Unfortunately these simulators are non-iterative, i.e., different simulations of the same circuit with different MOS models are completely independent simulations. In other words, the result of simulating a circuit using one type of MOS model cannot be exploited when the circuit is to be resimulated using another type of MOS model. Hence, the use of highly complicated model in the simulation of large circuits is frequently avoided due to its cost. In contrast, due to the iterative nature of the WR Method, we can take advantage of various existing models to increase the speed of simulation of circuits that use complicated models. The basic idea is to use the simple models in

the first few iterations and switch to more complicated models later when more accuracy is needed. This adaptive use of different models in the simulation actually corresponds to the concept of the *non-stationary* WR algorithm described in the previous chapter.

For the sake of simplicity, we shall assume that all capacitors in the circuit are linear and that there are only $N$ models for describing the drain-source characteristic of an MOS device. Let these models be ordered in terms of their computational complexities. By using Nodal Analysis formulation, the circuit equations of a circuit containing $n$ unknown node voltages can be written as

$$C\dot{v} + q^j(v, u) = 0 ; \qquad v(0) = V \tag{6.9}$$

where $C \in \mathbf{R}^{n\times n}$ is the capacitance matrix, $v \in \mathbf{R}^n$ is the vector of the unknown node voltages, $u \in \mathbf{R}^r$ is the input vector and $q^j : \mathbf{R}^n\times\mathbf{R}^r \to \mathbf{R}^n$ is the function associated with the $j$-th MOS model. Let

$$C = L + D + U \tag{6.10}$$

where $L$ is a strictly lower triangular matrix, $D$ is a diagonal matrix and $U$ is a strictly upper triangular matrix. Let $q^j_{GS} : \mathbf{R}^n\times\mathbf{R}^n\times\mathbf{R}^r \to \mathbf{R}^n$ be a vector valued function whose $i$-th component is defined as

$$q^j_{GS_i}(v, \tilde{v}, u) = q^j_i(v_1, \ldots, v_i, \tilde{v}_{i+1}, \ldots, \tilde{v}_n, u) \tag{6.11}$$

Let $\varepsilon_j$ ; $j = 1,2,\ldots,N$ be a predefined[1] sequence of positive numbers in which $\varepsilon_j$ specifies the simulation accuracy associated with the $j$-th MOS model. Then a WR algorithm that uses these models adaptively can have the following form.

---

[1] We assume that these simulation accuracy parameters are given either directly by the user or automatically by the simulator. For example, if the user wants to use only the first model, he can specify $\varepsilon_1 = 0$.

**Algorithm 6.3** (WR Algorithm with Adaptive MOS Models)

Step 1: Set $k = 1$, $j = 1$ and $v^0(t) = V$ for all $t \in [0,T]$.

Step 2: Solve for $v^k(t)$ ; $t \in [0,T]$ from

$$(L+D)\dot{v}^k + U\dot{v}^{k-1} + q^j_{CS}(v^k, v^{k-1}, u) = 0 \; ; \qquad v^k(0) = V.$$

Step 3: ( Accuracy test )

If ( $\max_{t \in [0,T]} \|v^k(t) - v^{k-1}(t)\| \leq \varepsilon_j$ and $j < N$ ), then set $j = j + 1$.

Step 4: Set $k = k + 1$ and go to step 2. ■

Assuming that $\varepsilon_j > 0$ for all $j$, then it is intuitively clear that this algorithm will converge to the solution of the circuit with the most accurate model. This is because when the algorithm stays with any model, it will tend to converge to the solution associated with that model and hence will pass the accuracy test in a finite number of iterations.

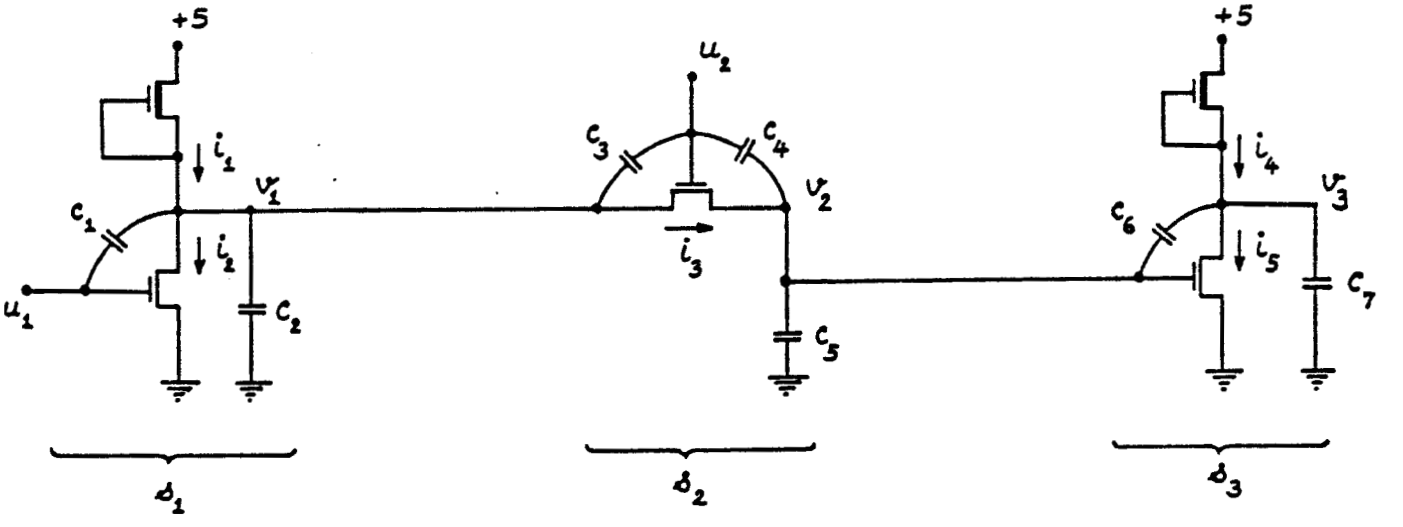



Fig. 6.1

An MOS dynamic shift register

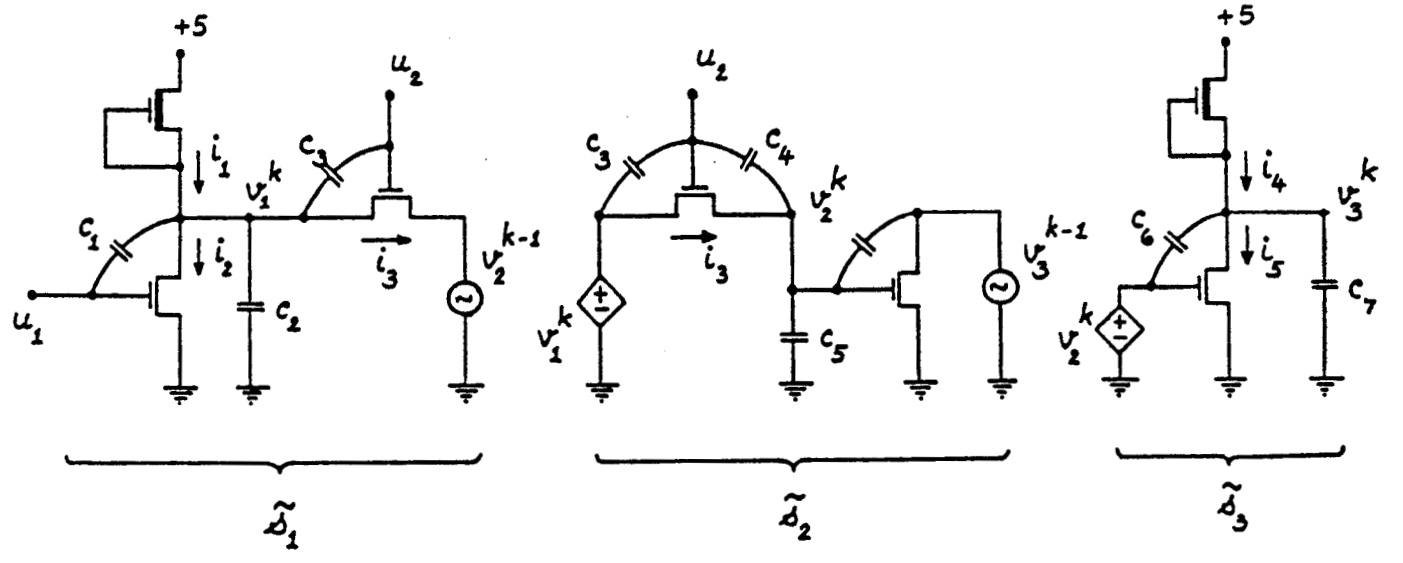



Fig. 6.2

The relaxation decomposition of the circuit in Fig. 6.1

at the $k$-th iteration of Algorithm 6.1

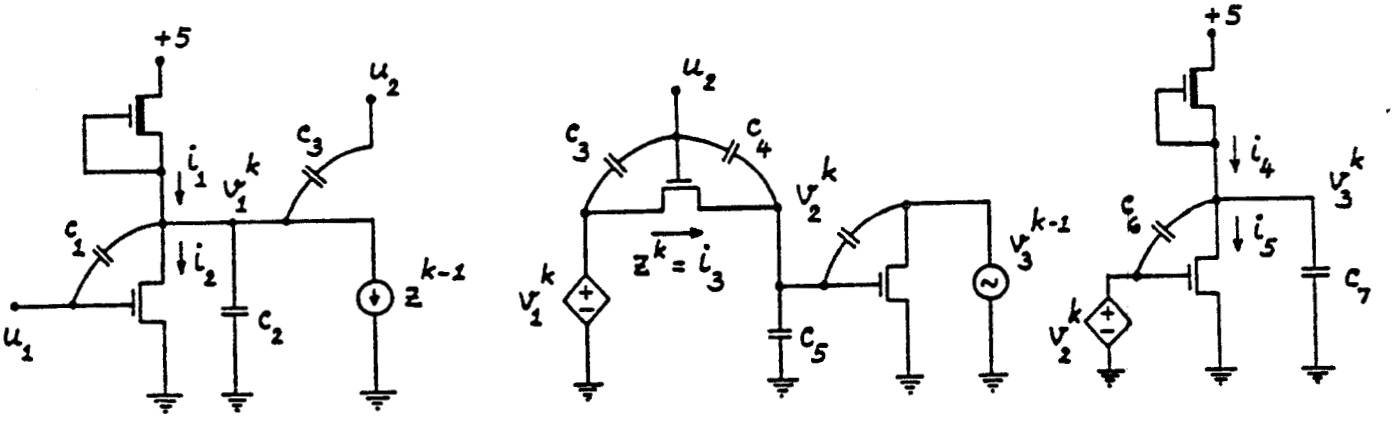



**Fig. 6.3**

The relaxation decomposition of the circuit in Fig. 6.1

at the $k$-th iteration of Algorithm 6.2

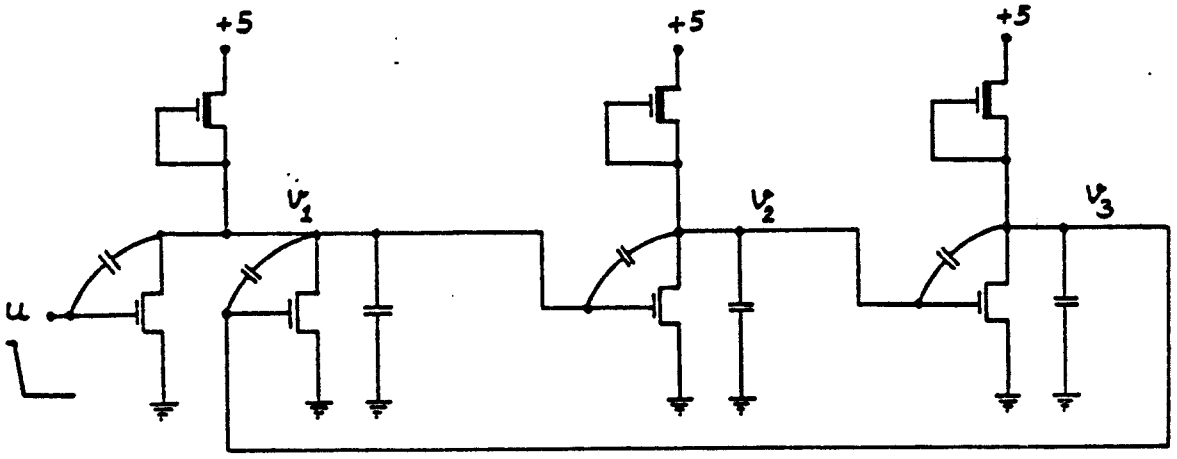



Fig. 6.4

An MOS ring oscillator

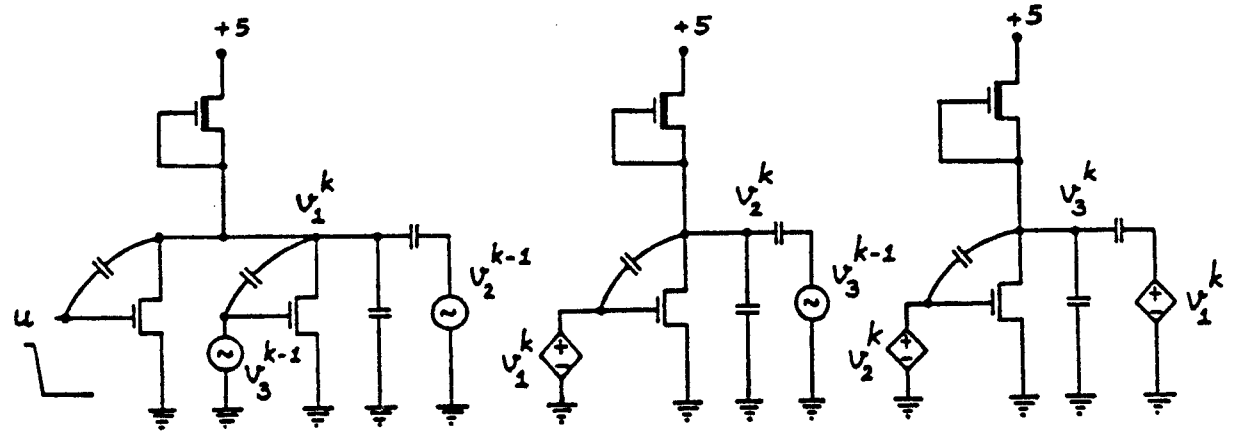



Fig. 6.5

The relaxation decomposition of the circuit in Fig. 6.4

at the $k$-th iteration of Algorithm 6.1

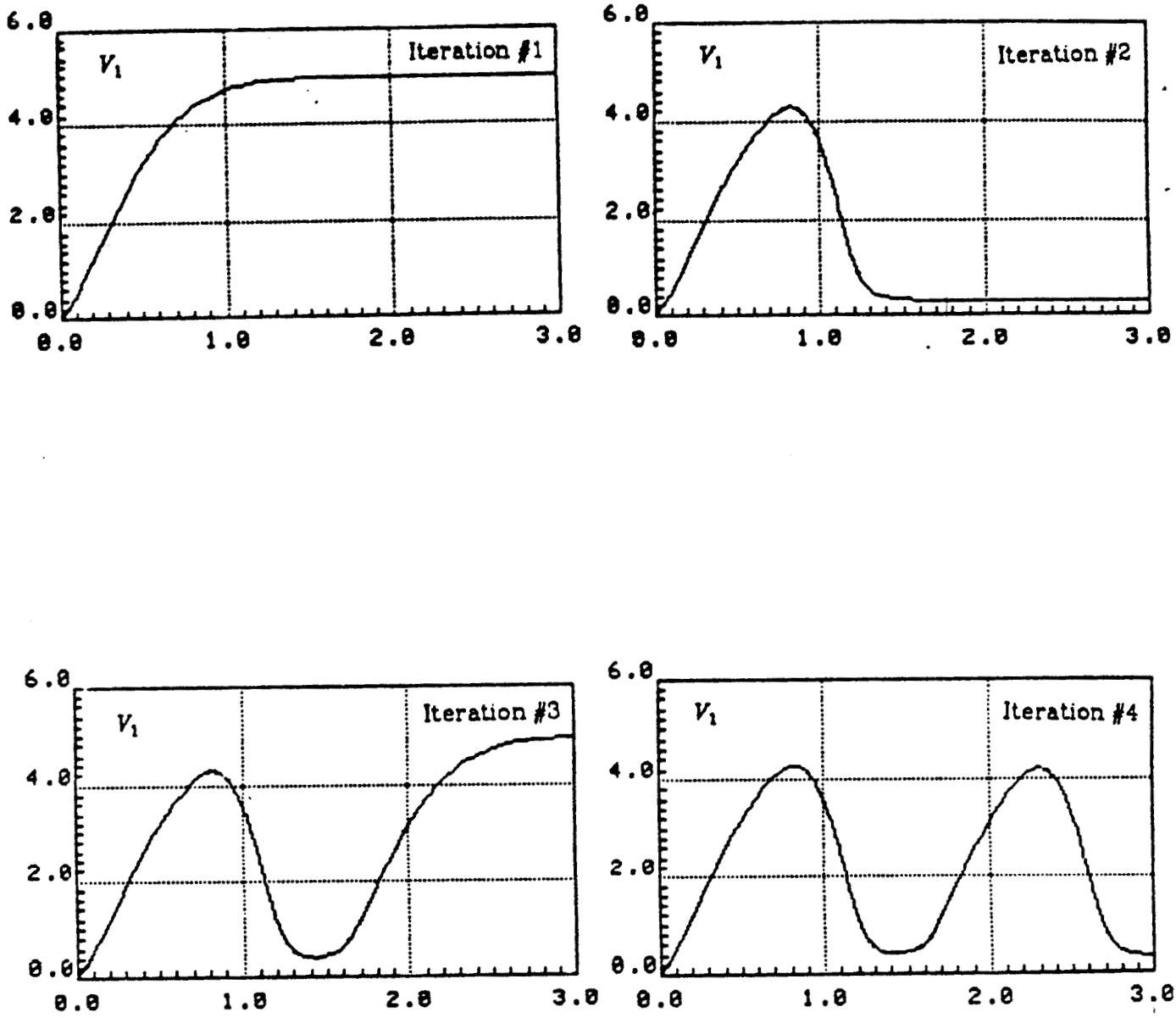



**Fig. 6.6**

Waveforms at various iterations of Algorithm 6.1 applied to the circuit in Fig. 6.4, assuming that $v_3^0(\cdot) = 0$

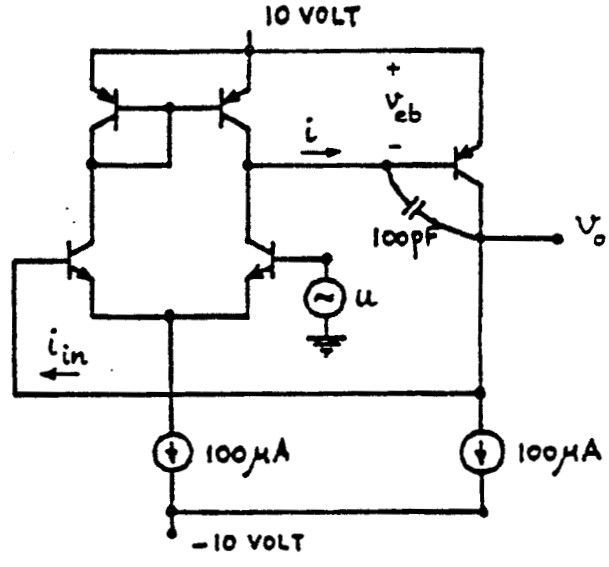



Fig. 6.7a

A simplified bipolar transistor operational amplifier

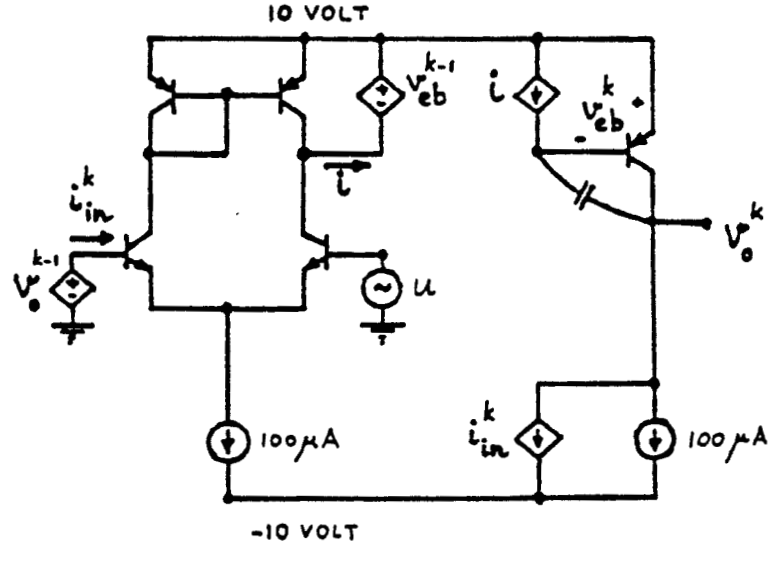



Fig. 6.7b

A relaxation decomposition of the circuit in Fig. 6.7a

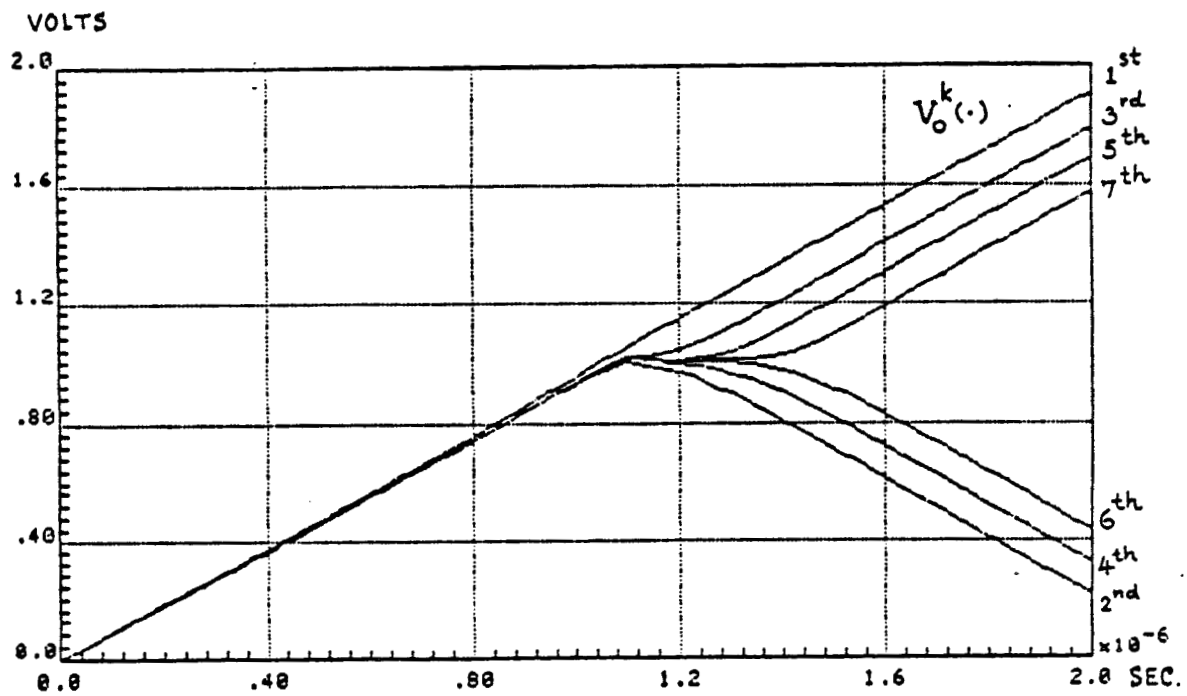



Fig. 6.8

Waveforms at various iterations of the solution of the circuit in Fig. 6.7b

# Chapter 7

# RELAX: An Experimental MOS Digital Circuit Simulator

In this chapter, we describe the basic numerical techniques used in RELAX: an experimental circuit simulator which implements a WR algorithm and is specially designed to simulate MOS digital circuits. In particular, a few important techniques to improve the convergence property and the execution speed of simulation will be described, namely the *scheduling* algorithm and techniques to exploit the *latency* and the *partial waveform convergence*.

## 7.1. Basic Algorithms in RELAX.

RELAX implements a modified version of the WR Algorithm 6.1, as described in the previous chapter. These modifications are described as follows.

1) Rather than having strictly one equation per each decomposed system, RELAX allows each partitioned subsystem to have more than one equation so that each subsystem corresponds to a physical digital subcircuit, e.g. NOR, NAND, FLIP-FLOP etc. In fact, the choice of the decomposition is dictated by its input language, i.e., the user specifies his digital circuit as an interconnection of several subcircuits.

2) Each decomposed subcircuit is solved by using conventional simulation techniques as described in Chapter 2. The Backward Euler integration method with variable timesteps is used to discretize the differential equations associated with the subcircuit and the Newton-Raphson method is

used to solve the nonlinear algebraic equations resulting from the discretization. Since the number of unknown variable associated with a subcircuit is usually small, the linear equation solver used by the Newton-Raphson method is implemented by using the standard full matrix techniques rather than using the sparse matrix techniques. Note that in RELAX each subcircuit is analyzed independently from $t = 0$ to $t = T$ using its own timestep sequence controlled by the integration method, whereas in a standard circuit simulator the entire circuit is analyzed from $t = 0$ to $t = T$ using only one common timestep sequence. In RELAX, the timestep sequence of one subcircuit is usually different from the others but contains, in general, a smaller number of timesteps than that used in a standard circuit simulator for analyzing the same circuit.

3) The first iteration of RELAX is essentially the first iteration of Algorithm 6.2. But after that RELAX switches back to use Algorithm 6.1 for the rest of the relaxation iteration. That is, in the first WR iteration of RELAX, the drain currents of the pass transisters do not contribute any loading effect on the subcircuits to which they are connected. This is done because, in the first iteration when all initial guesses are constant waveforms, a pass transistor can be driven continuously into its conductive region and may adversely effect the speed of convergence if its current is treated as a load of the other subcircuit. Hence, strictly speaking, the first iteration in RELAX is used to generate a good initial guess for the actual WR algorithm.

## 7.2. Scheduling Algorithm.

The order according to which each subcircuit is processed is determined in RELAX prior to starting the WR iteration by a subroutine called the "scheduler". Although it has been shown in Theorem 6.1 that *scheduling* is not necessary to

guarantee convergence of the iteration, it does have an impact on the speed of convergence. Assume that the circuit consists of unidirectional subcircuits with no feedback path. If the subcircuits are processed according to the *flow of signals* in the circuit, the WR algorithm used in RELAX will converge in just two iterations (actually the second iteration is needed only to verify that the convergence has been obtained). For MOS digital circuits which contain almost unidirectional subcircuits, it is intuitive that the convergence of the WR algorithm will be achieved more rapidly if the subcircuits are processed according to the flow of signals in the circuit. The scheduler traces the flow of signals through the circuit and orders the processing of subcircuits accordingly. To be able to trace the flow of signals, the scheduler requires the user to specify the flow of signals through each subcircuit by partitioning the terminals of the subcircuit into the *input* and the *output terminals*. In general, a designer can easily specify what the flow of the signals is intended to be even in a subcircuit which is not unidirectional such as a transmission gate or a subcircuit containing floating capacitors between its input and output terminals. For example, the input circuit description of the circuit shown in Fig. 6.1 could be described as shown in Fig. 7.1. Note that the analysis algorithm in RELAX will indeed take into account the bidirectional effects correctly. To describe the algorithm used by the scheduler, we need the following definition.

**Definition 7.1** A subcircuit $s_2$ is said to be a *fanout* of a subcircuit $s_1$ if an input terminal of $s_2$ is connected to an output terminal of $s_1$, i.e., an output of $s_1$ is fed as an input to $s_2$. ∎

Before stating the scheduling algorithm, we point out that all real input signals to the circuit are considered by the scheduler to be contained in a special subcircuit called the "*source*" subcircuit which is essentially a subcircuit with only output terminals. The algorithm traces the flow of signals from the source

subcircuit through the circuit by using the fanout information of each subcircuit. When there is a logic feedback loop, the loop is temporary opened. The details of the algorithm is as follows.

**Scheduling algorithm.**

Comment: $X$ is an ordered set of subcircuits. At the completion of the algorithm, $X$ contains all the subcircuits and the order in which each subcircuit is placed in $X$ is the order in which it is processed by RELAX. $X$ is called the *scheduling table*.

Start: Set $X$ = {source subcircuit} and $Y$ = {fanouts of the source subcircuit}.

LOOP: Set $Z = \varphi$, i.e., clear the temporary set $Z$.

For each subcircuit $s$ in $Y$

Begin

If ( all inputs of $s$ come from the outputs of subcircuits in $X$ ) then

Begin

Delete $s$ from $Y$ and add it to $X$.

Include in $Z$ the fanouts of $s$ which are not already in $X$, $Y$ or $Z$.

End

End

If ( $Z$ is not empty ) then include $Z$ in $Y$ and go to LOOP.

Else if ( $Y$ is empty ) then stop

Else Begin ( comment: there's a feedback loop )

Select a subcircuit $s$ from $Y$.

Delete $s$ from $Y$ and add it to $X$.

Include in $Y$ the fanouts of $s$ which are not already in $X$ or $Y$.

Go to LOOP.

End. ∎

For example, the set $X$ produced by the scheduler applied to the circuit of Fig. 7.2 is { source, $s_1$, $s_5$, $s_4$, $s_2$, $s_3$ } and the set $X$ produced by the scheduler applied to the circuit of Fig. 7.3 is { source, $s_2$, $s_3$, $s_1$}. Note that this scheduling process is carried out only once before starting the WR iteration.

## 7.3. Latency and Partial Waveform Convergence.

In addition to the modifications described in Section 7.1, RELAX incorporates two important techniques to speed up the process of analyzing a subcircuit. The key idea is to bypass the analysis of a subcircuit for certain time intervals without losing accuracy by exploiting the information obtained from previous timepoints and/or from previous iterations. Similar techniques have been used in other simulators. For example, SPICE uses a bypass technique [3] in its Newton-Raphson iteration. When a subvector of the vector of the unknown variables does not change its value significantly in the previous two NR iterations, the part of the Jacobian matrix associated with the subvector is not recomputed. SPLICE, on the other hand, uses an event scheduling technique [5] by which a subcircuit is not scheduled to be analyzed at an analysis timepoint if it is found to be inactive at that timepoint.

The two techniques used in RELAX are discussed by showing their applications to the analysis of the subcircuit $s_1$ of the circuit shown in Fig. 7.4a which is a schematic diagram of the circuit in Fig. 6.1. We denote the output voltages of $s_1$ and $s_2$ at the $k$-th WR iteration by $v_1^k$ and $v_2^k$ respectively.

The first technique is based on the *latency* of $s_1$ and is similar to the technique described in [7]. According to Section 7.1, $s_1$ is analyzed in the first iteration with no loading effect from $s_2$. After it has been analyzed for a few timepoints, its output voltage $v_1^1$ is found to be (almost) constant with time, i.e., $\dot{v}_1^1(0.01) \approx 0$ (see Fig. 7.4b). Since the input $u_1$ of $s_1$ is also constant during the

interval [0.01, 1.9], $v_1^1$ will also remain constant throughout the interval [0.01, 1.9]. The subcircuit $s_1$ is then said to be "*latent*" in the first iteration during the interval [0.01, 1.9] and its analysis during this interval is bypassed. From Fig. 7.4b, $s_1$ is latent again in the interval [2.15, 3]. Note that, according to Section 7.1, the check for the latency of $s_1$ after the first iteration will include $u_2$ and $v_2$ as well as $u_1$ since they can effect the value of $v_1$. For most digital circuits, the latency intervals of a subcircuit usually cover a large portion of the entire simulation time interval $[0,T]$ and hence the implementation of this technique can provide a considerable saving of computing time as shown in Table 7.1.

The second technique is a unique feature of the WR algorithm. It is based on the *partial convergence* of a waveform during the previous two WR iterations. We introduce it by using the example of Fig. 7.4 as follows. After the first two iterations, we observe that the values of $v_1^1$ and $v_1^2$ during the interval [1.7, 3.0] do not differ significantly (see Fig. 7.4b, 7.4c and 7.4e), i.e., the sequence of waveforms of $v_1$ seems to converge in this interval after two iterations. In the third iteration, shown in Fig. 7.4d, $s_1$ is analyzed from $t = 0$ to $t = 1.8$ and $v_1^3(1.8)$ is found to be almost the same as $v_1^2(1.8)$. Moreover, during the interval [1.8, 3], the value of $v_2^2$ which effects the value of $v_1^3$ also does not differ significantly from the values of $v_2^1$ ( which effects the value of $v_1^2$). Hence the value of $v_1^3$ during the interval [1.8, 3] should remain the same as $v_1^2$ and the analysis of $s_1$ during this interval in the third iteration will be bypassed. This technique can provide a considerable saving of computing time as shown in Table 7.1 since the intervals of convergence can cover a large portion of the entire simulation time interval $[0,T]$, especially in the last few iterations. Note that the subcircuit need not be latent during the intervals of convergence although the overlapping of these intervals with the latency intervals is possible.

```
Comment: Description of the circuit shown in Fig. 6.1.
s1  inverter              input = node1          output = node2
s2  inverter              input = node3          output = node4
s3  transmission-gate     input = node5, node2   output = node3

Comment: Description of the models of MOS transistors.
Model ENHANCE    NMOS {MOS parameters such as threshold
                       voltages, transconductance, etc.}
Model DEPLETION  NMOS {MOS parameters}

Comment: The connectivity of each MOS device is described as
       : MOS-name     drain-node  gate-node  source-node  body-node &
       :              MOS-model-name  width      length .

Comment: Description of transmission gate.
subcircuit transmission-gate input = source, gate output = drain
   MOS1 drain gate source GROUND ENHANCE width = 1μ length = 1μ
ends

Comment: Description of an inverter.
subcircuit inverter input = A output = Ā
   MOSload VDD Ā Ā GROUND DEPLETION width = 1μ length = 1μ
   MOSdriver Ā A GROUND GROUND ENHANCE width = 4μ length = 1μ
ends
```

**Fig. 7.1**

Example of an input circuit description for RELAX.

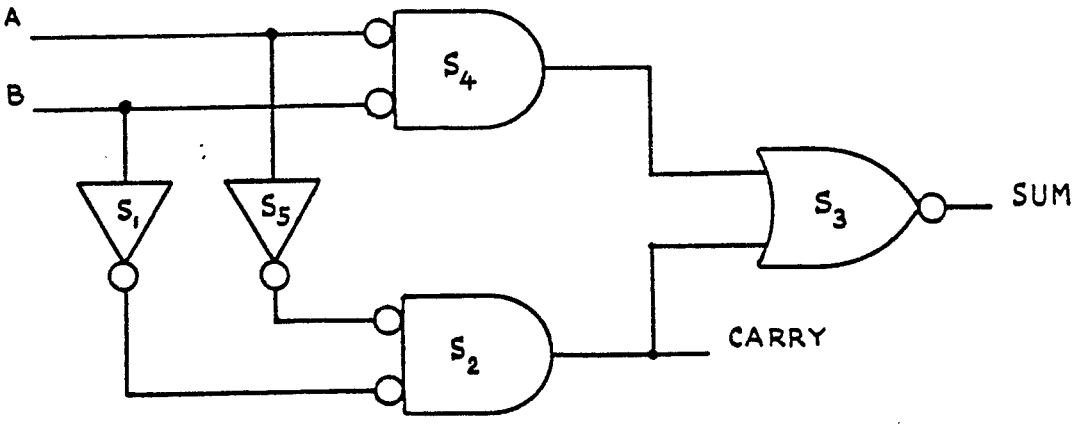



**Fig. 7.2**

A half adder

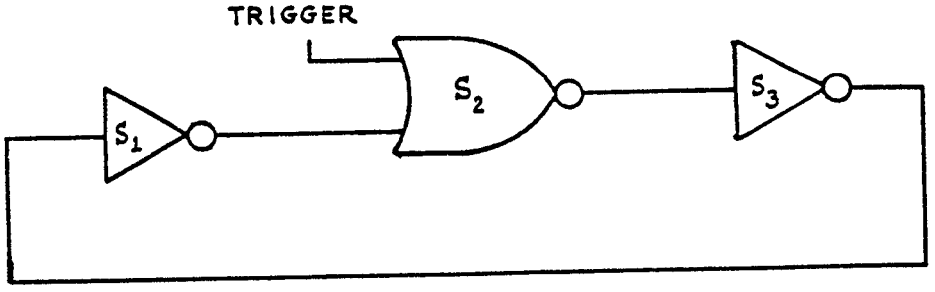
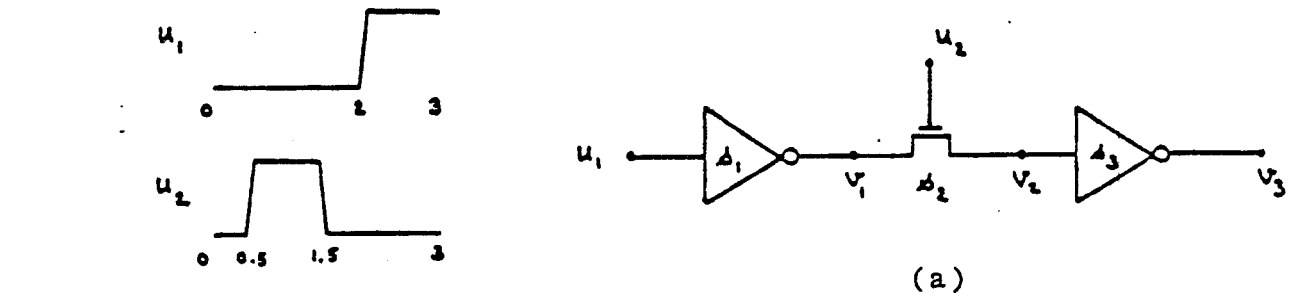
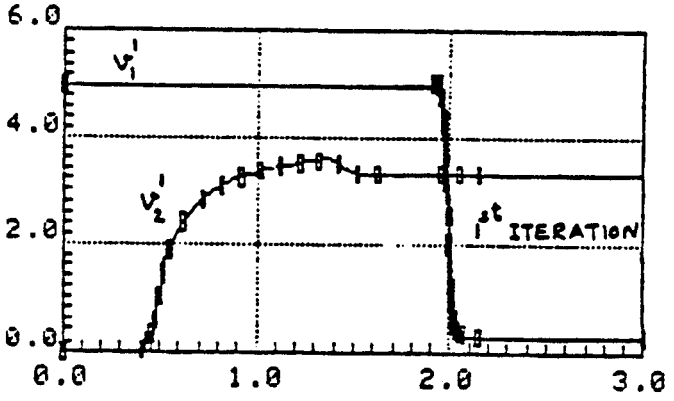



**Fig. 7.3**

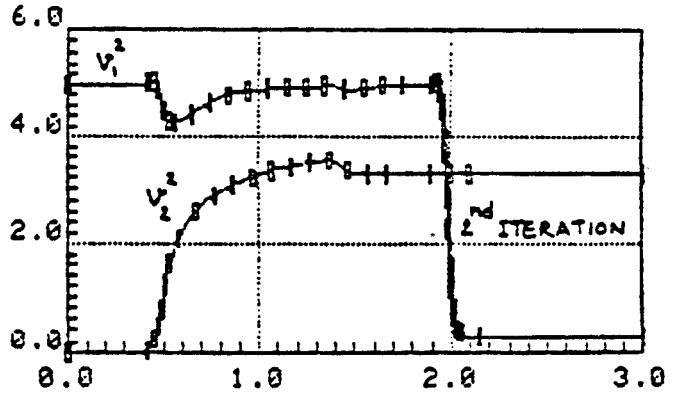
A ring oscillator



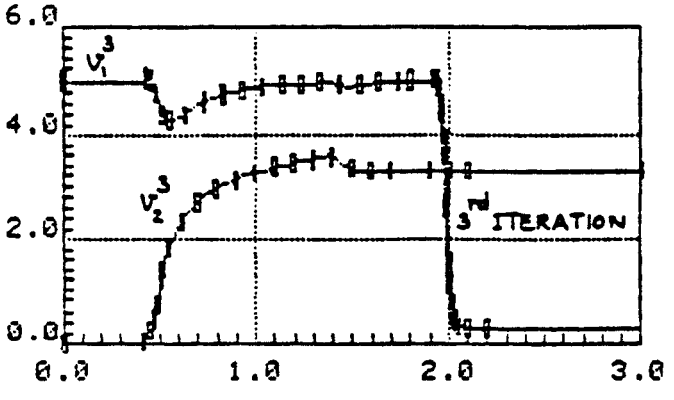

(a)



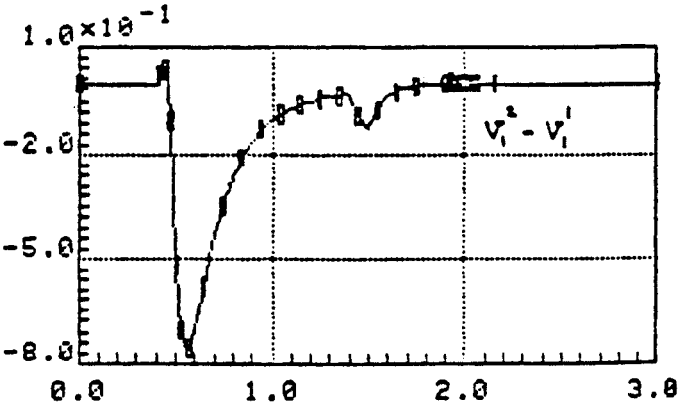

(b)




(c)




(d)




(e)

**Fig. 7.4**

A dynamic shift register and the waveforms at various iterations of the WR algorithm used in RELAX.

| iteration # | CPU time ( seconds ) | | | |
|---|---|---|---|---|
| | case 1 | case 2 | case 3 | case 4 |
| 1 | 0.363 | 0.255 | 0.360 | 0.258 |
| 2 | 0.824 | 0.704 | 0.814 | 0.695 |
| 3 | 0.821 | 0.705 | 0.242 | 0.238 |
| 4 | 0.828 | 0.704 | 0.151 | 0.104 |
| 5 | 0.832 | 0.695 | 0.097 | 0.016 |
| Total | 3.668 | 3.063 | 1.664 | 1.311 |

**Table 7.1**

Comparison of CPU times used by RELAX for analyzing the circuit of Fig. 7.4a with and without the latency and the partial waveform convergence techniques.

case 1: without the latency and the partial waveform convergence techniques.

case 2: with only the latency technique.

case 3: with only the partial waveform convergence technique.

case 4: with both the latency and the partial waveform convergence techniques.

# Chapter 8

# Organization of RELAX

RELAX is written in FORTRAN and runs in an interactive mode. The main routine of RELAX acts as an interface between the user and the processors (i.e., subroutines). It interprets the input commands and activates the corresponding internal processors (implemented by subroutine calls). Some typical RELAX commands are for

1) reading the description of the circuit from an external file,

2) continuing the execution of the WR iteration,

3) setting the accuracy of the analysis,

4) monitoring the waveforms at each iteration and

5) leaving the program.

The organization of the main internal processors of RELAX is shown in Fig. 8.1. We now describe the function of each processor.

The *input circuit processor* reads the description of the circuit from the specified external file and stores it in a compact form in an internal array. As mentioned in the previous chapter, the circuit must be entered as an interconnection of subcircuits whose input and output terminals are clearly specified. The *scheduler* then reads the internal array produced by the input circuit processor and generates a *fanout table* for each subcircuit according to Definition 7.1. Then it executes the scheduling algorithm described in the previous chapter to produce a *scheduling table* that gives the order in which each subcircuit will be processed in the WR iteration. Both the input circuit processor and the scheduler are actually the preprocessing steps for the WR iteration since they are performed only once for the circuit to be analyzed.

The analysis of a subcircuit using the WR iteration is implemented in RELAX as a two-phase process: the *setup phase* performed by the *intermediate code generator* and the *analysis phase* performed by the *subcircuit analyzer*. The intermediate code generator reads the description of the subcircuit and its fanouts (obtained from the input circuit processor and the scheduler) and generates an intermediate code. This intermediate code is used by the subcircuit analyzer to analyze the subcircuit from $t = 0$ to $t = T$, where $T$ is the user-specified simulation time interval. The subcircuit analyzer consists of several subroutines implementing the Backward Euler integration method, the Newton-Raphson method, the linear equation solver and the two techniques for speeding up the analysis discussed in the previous chapter.

The output and internal voltages of the subcircuit at the sequence of timepoints used by the subcircuit analyzer as well as the sequence of timepoints are stored in an *internal waveform storage* which stores the discretized waveforms associated with all nodes in the circuit. To analyze a subcircuit at a timepoint, say $t_1$, the subcircuit analyzer has to know the values of its input voltages and the voltages associated with its fanouts at time $t_1$. However, since the sequence of timepoints for analyzing one subcircuit may be different from the others, $t_1$ may not coincide with any of the timepoints associated with the required voltages in the waveform storage. Hence an interpolation has to be performed to obtain the required values when such case arises. In RELAX, the subcircuit analyzer obtains the values of the input voltages and the voltages of the fanouts of a subcircuit from a utility subroutine, called the *interpolator*, which reads the waveform storage and performs the interpolation (if necessary) to get the values at the specified timepoint.

In addition, the interpolator reads the waveform storage and performs the interpolation (if necessary) to get the values of the output and internal voltages of the subcircuit at the specified timepoint in the previous iteration. The differences between these values and the corresponding values in the current iteration are also stored in the waveform storage. These differences will be used in the next iteration by the routine implementing partial waveform convergence technique described in the previous chapter. At the end of the analysis of the subcircuit, the discretized waveforms associated with the subcircuit in the previous iteration are no longer needed and the storage occupied by them can be reused.

At present, RELAX is still in an experimental stage. It can handle MOS digital circuits containing NOR gates, NAND gates, transmission gates, multiplexers (or banks of transmission gates whose outputs are connected together), super buffers and cross-coupled NOR gates (or flip-flops). It uses the Schichmann-Hodges model [9] (or the level 1 MOS model used in SPICE2) for the MOS device. All the computations are performed in double precision and the results are also stored in double precision. Although RELAX code is rather small, approximately 4000 FORTRAN lines, it requires a considerably large amount of storage for the waveforms, especially when large circuits are analyzed. For an MOS circuit containing 1000 nodes with 100 analysis timepoints per node, the waveform storage is required to store approximately 3×1000×1000 floating point numbers (corresponding to 2.4 megabytes if each number is stored in 64 bits). Further enhancements of RELAX are

1) Capability of handling user-defined subcircuits.

2) Provision of more accurate MOS models. The user, if desired, can choose to use a simplified MOS model in the first few iterations for a fast analysis and switches to a more accurate model in the later iterations.

## 8.1. Look-ahead Storage Buffering Scheme

One of the drawback of RELAX is the fact that it has to store all the waveforms at the current iteration and reuse them in the next iteration. For large circuits which usually require large simulation time intervals, the amount of storage required to store the waveforms can be extremely large and makes it infeasible to store all the waveforms in the primary storage of a computer. However, as we have mentioned earlier in Chapter 6, the circuit equations of most practical circuits are usually sparsa. This means that not all waveforms are required in analyzing any particular subcircuit. Furthermore, the order of analyzing subcircuits is predetermined and stored in the "scheduling table". Therefore, by exploiting these two facts, it is possible to use a slow but large storage medium such as a disk to provide additional storage for the waveforms without sacrificing much of the execution speed due to its slow access time. This is achieved by a technique which we call the *look-ahead storage buffering scheme*[1] and is described as follows.

In the look-ahead storage buffering scheme, the amount of primary storage allocation for the waveforms is limited. When this storage is not enough to store all the waveforms, a secondary storage such as a disk is used to supply the additional storage needed. Since the access time of the secondary storage is much longer than that of the primary storage, the speed of the analysis of a subcircuit will be greatly reduced if the interpolator has to access the secondary storage directly in order to get the desired values. The buffering scheme is designed to cope with this situation and ensure that all the waveforms associated with the analysis of a subcircuit already reside in the primary storage prior to the beginning of the analysis of the subcircuit. This is achieved by using the following algorithm.

[1] Currently, this scheme has not yet been implemented in RELAX.

**Algorithm 8.1** *( Simplified look-ahead storage buffering algorithm )*

Comment: $s$ denotes the subcircuit currently being analyzed and $S$ denotes the set of the waveforms required in the analysis of $s$. For the sake of simplicity, we assume that the storage of the waveforms associated with the output and internal voltages of $s$ in the previous iteration is reused by the corresponding waveforms in the current iteration. We also assume that $S$ is in the primary storage.

LOOP: From the scheduling table, determine the next subcircuit to be analyzed.

Set $\tilde{S}$ = { waveforms required in the analysis of the next subcircuit }.

If ( $\tilde{S}$ is in the primary storage ) then go to WAIT

Else Begin

Set $Y$ = { waveforms in $\tilde{S}$ which are in the secondary storage }.

Select a set $Z$ of the waveforms in the primary storage which are not in $S$ or $\tilde{S}$ such that the amount of storage occupied by $Z$ is larger or equal to that of $Y$.

Transfer $Z$ to the secondary storage.

Transfer $Y$ to the primary storage occupied by $Z$.

Go to WAIT.

End.

WAIT: Wait until the analysis of $s$ is finished.

Set $s$ = the next subcircuit and $S = \tilde{S}$.

Go to LOOP. ∎

**Remark:**

We can easily modify the above storage buffering algorithm to allow the algorithm looks ahead more than one subcircuit.

Fig. 8.2 illustrates the storage organization proposed by this algorithm and the main data associated with the algorithm. Note from the above algorithm that the look-ahead storage buffering process can be executed concurrently with the process of analyzing the subcircuit $s$ since they do not access the same storage locations. Therefore by using this scheme RELAX will be able to analyze large circuits without requiring a large amount of primary storage and without reducing its speed.

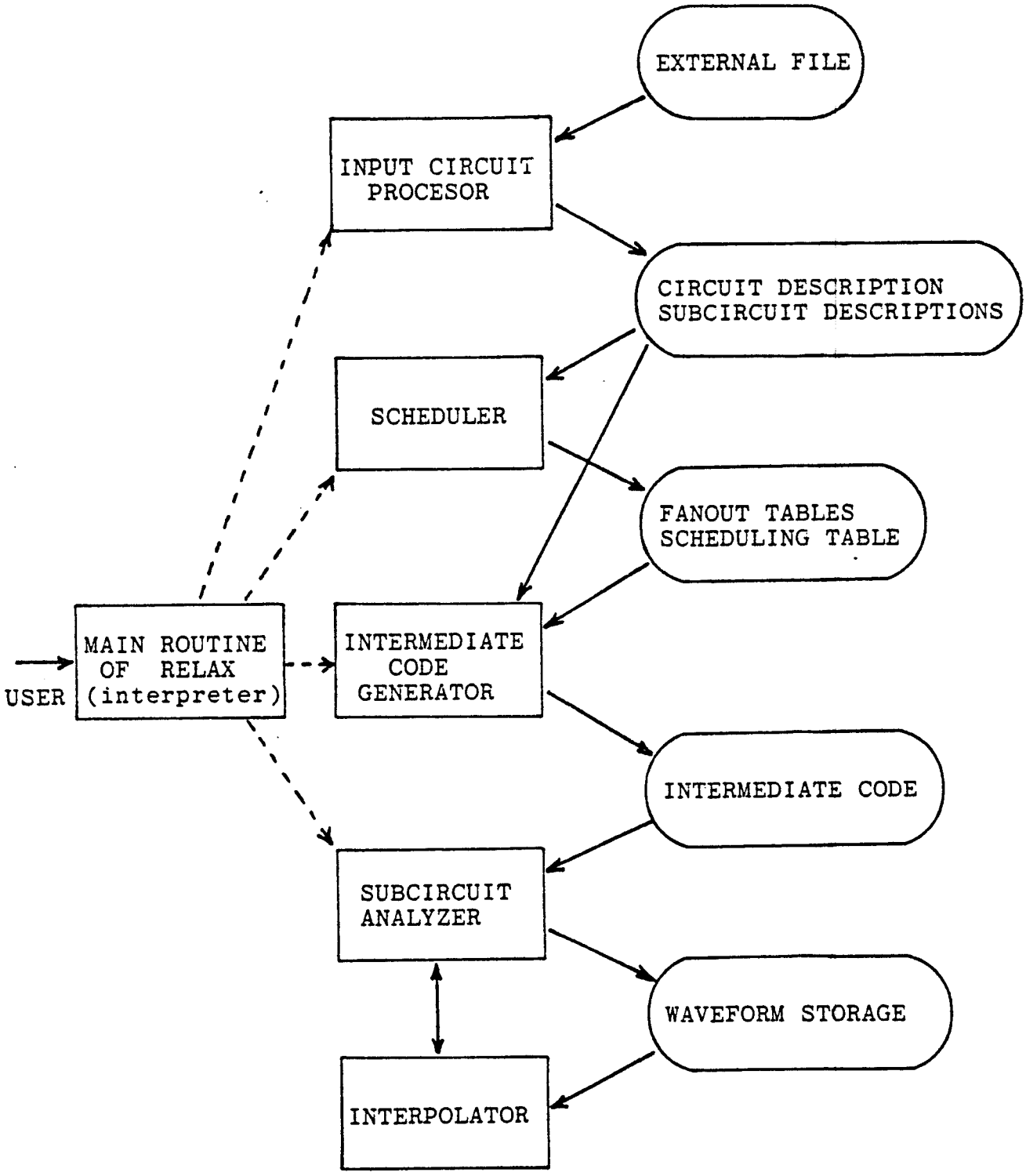



**Fig. 8.1**

Organization of RELAX

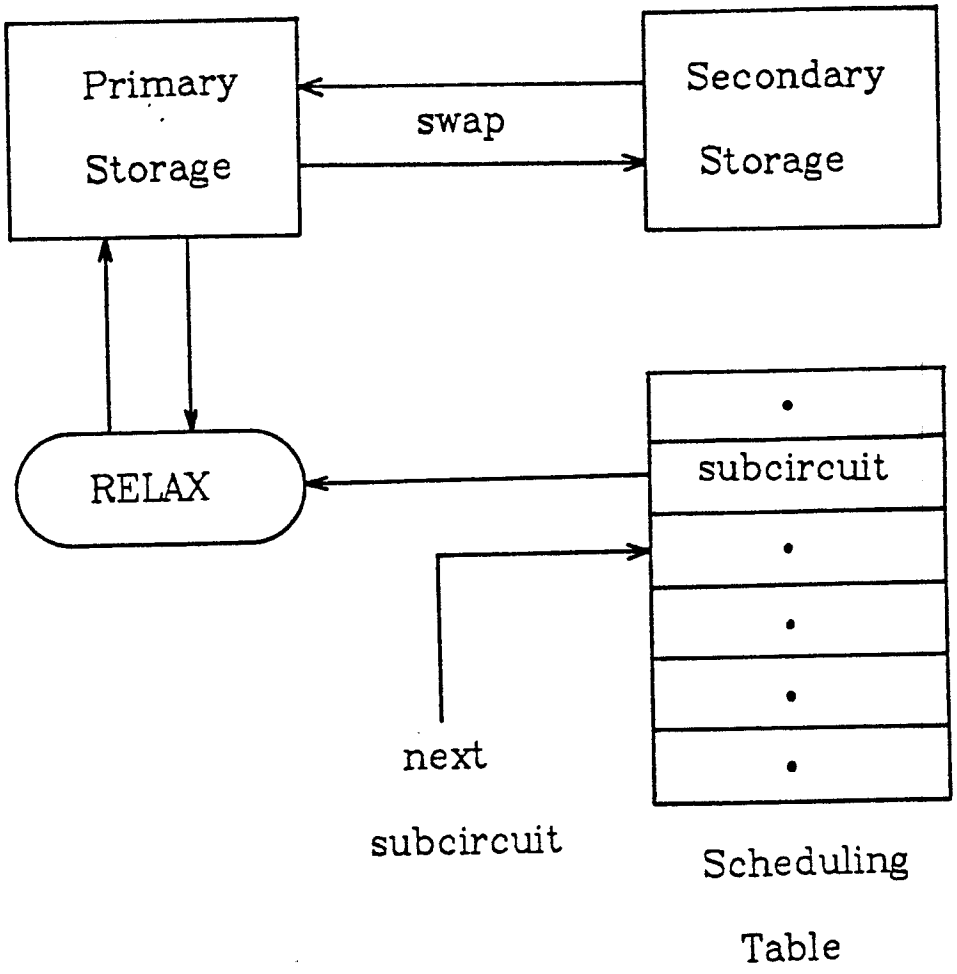



**Fig. 8.2**

Look-ahead storage buffering scheme

# Chapter 9

# Performance of RELAX

In this chapter, we describe the performance of RELAX and compare it with a standard circuit simulator: SPICE2. Since RELAX has not yet been fully developed and presently can handle very limited types of subcircuits and MOS models, the comparison serves only to show that the WR method is very suitable for simulating large scale digital integrated circuits. The two simulators use the same MOS model, i.e., the Schichman-Hodges model (or SPICE2 MOS model level = 1) with linear capacitors, so that the accuracy of RELAX can also be verified by using the outputs of SPICE as references. For RELAX, the specified convergence error for its WR iteration is 0.05 Volt.

The schematic diagrams of the MOS circuits being tested and their output waveforms obtained by RELAX and SPICE are shown in Fig. 9.1 through Fig. 9.5. For RELAX output waveforms, each rectangular mark denotes the computed value at every other two internal timepoints to illustrate the effect of the implemented latency technique. The two simulators run on a VAX 11/780 using UNIX[2] operating system. A comparison of the analysis time in CPU seconds spent by each simulator is given in Table 9.1. The tabulated CPU time for SPICE is the total CPU-seconds spent only by its transient analysis routine, i.e., they do not include the read-in, set-up and read-out phases. The tabulated CPU time for RELAX is the total CPU-seconds spent by the intermediate code generator, the subcircuit analyzer and the interpolator of the program as described in the previous chapter. The total number of iterations used by RELAX is also tabulated. It is clear from the figures and the table that RELAX can analyze MOS digital circuits at least one order of magnitude faster than SPICE while achieving the same accuracy.

---

[2] Unix is a trade mark of Bell Laboratory.

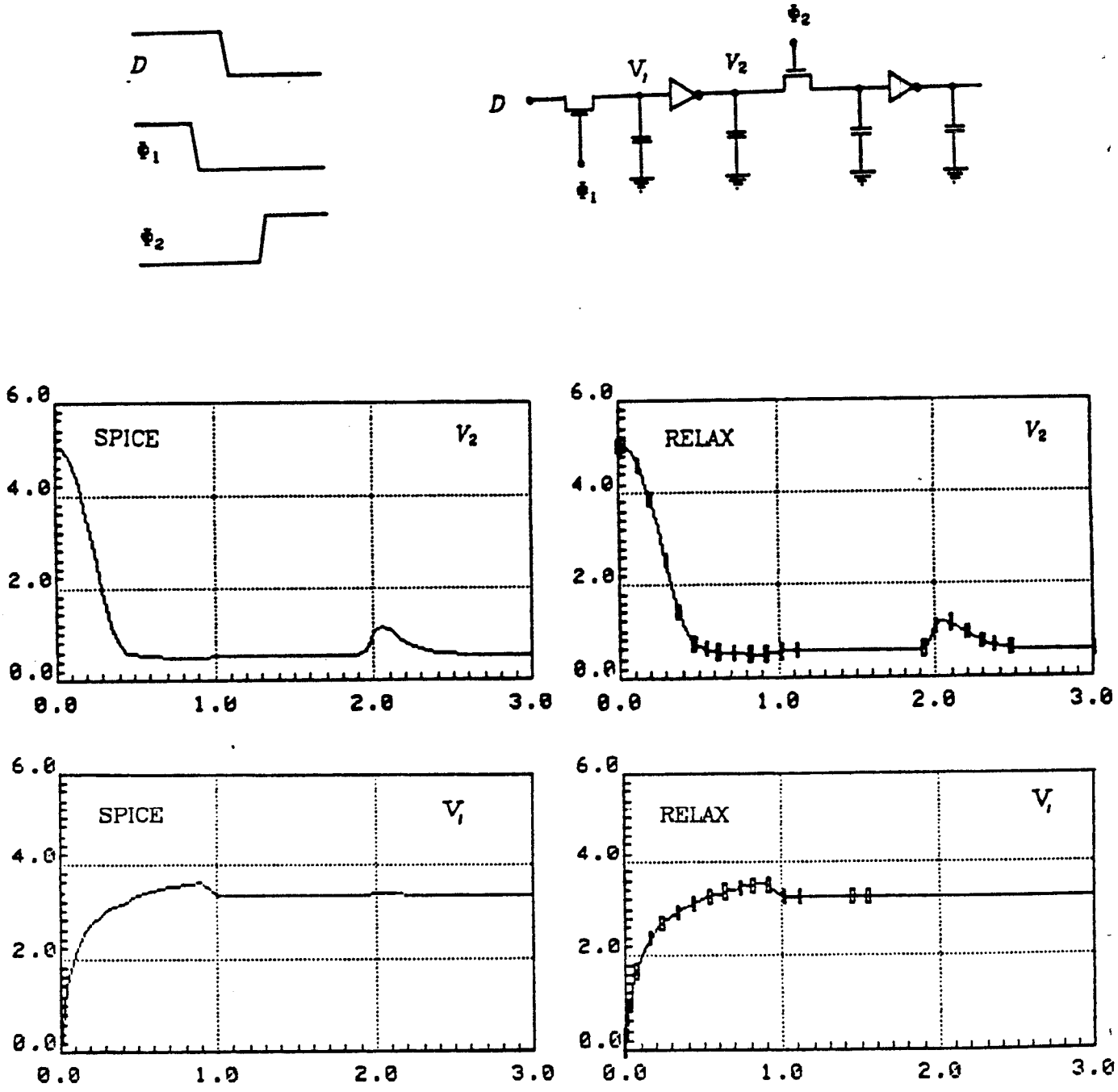
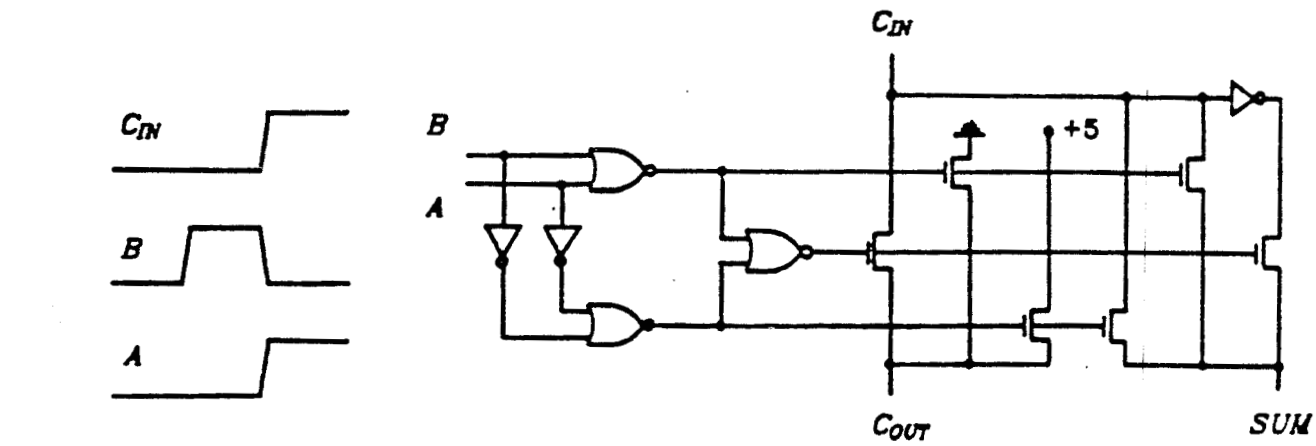



Fig. 9.1

A dynamic shift register



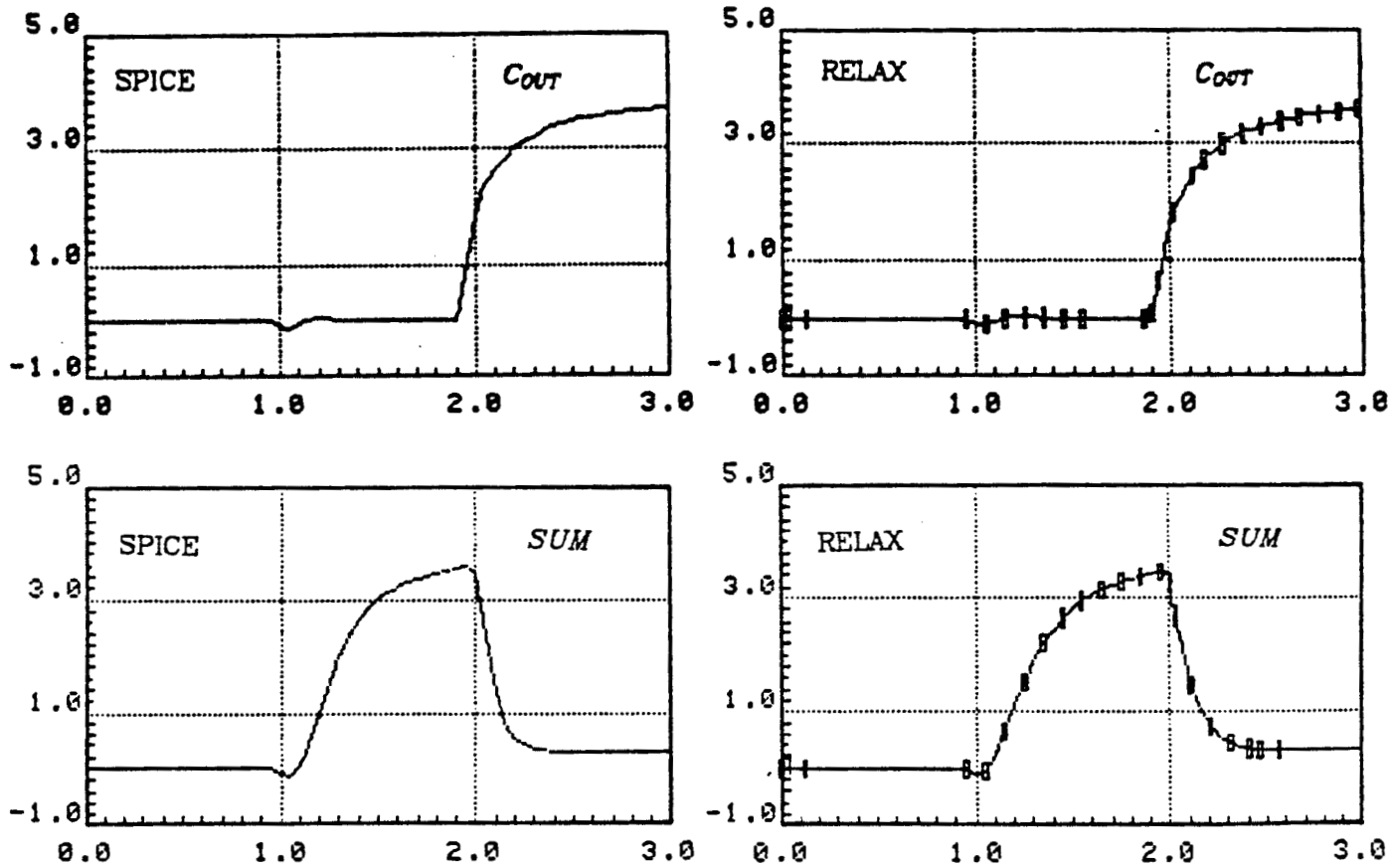
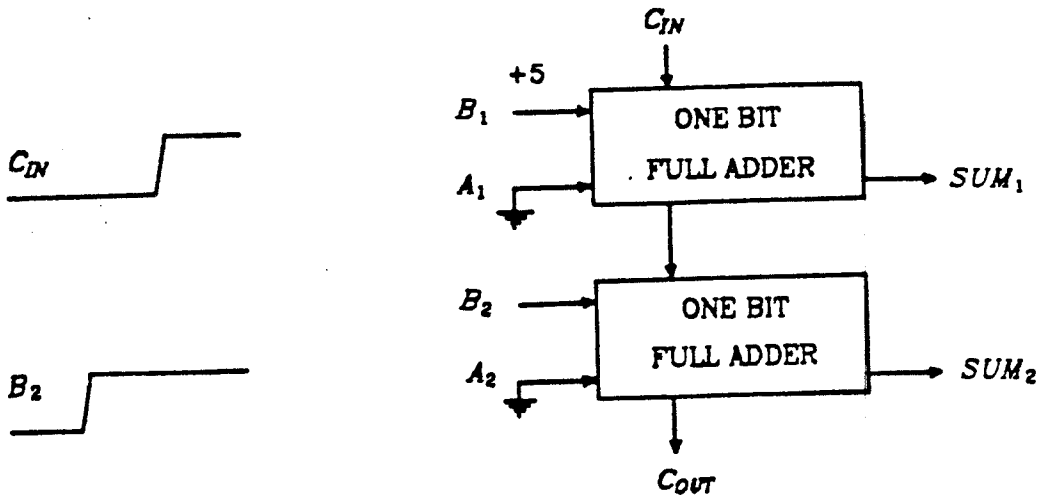
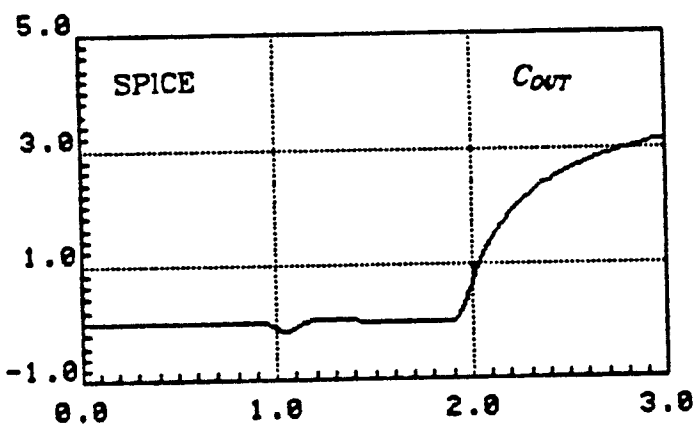
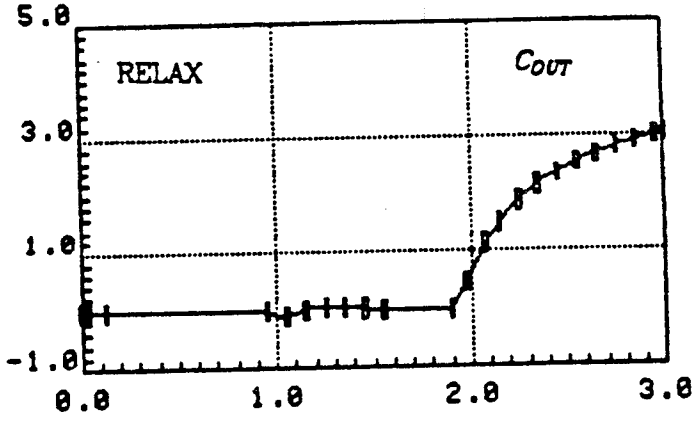



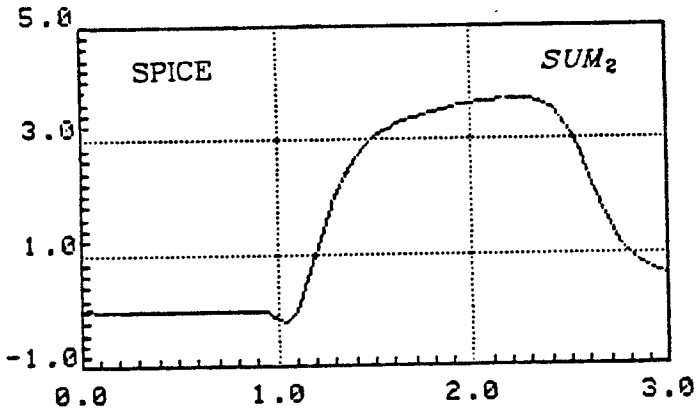

Fig. 9.2

A one-bit full adder using a pass transistor carry chain



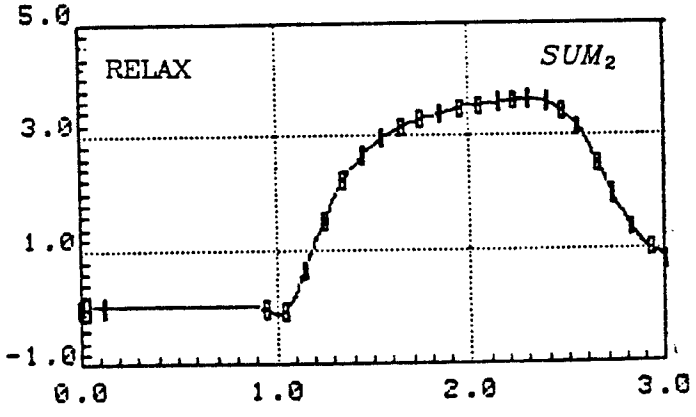




**Fig. 9.3**

A two-bits full adder using two one-bit full adders of Fig. 9.2

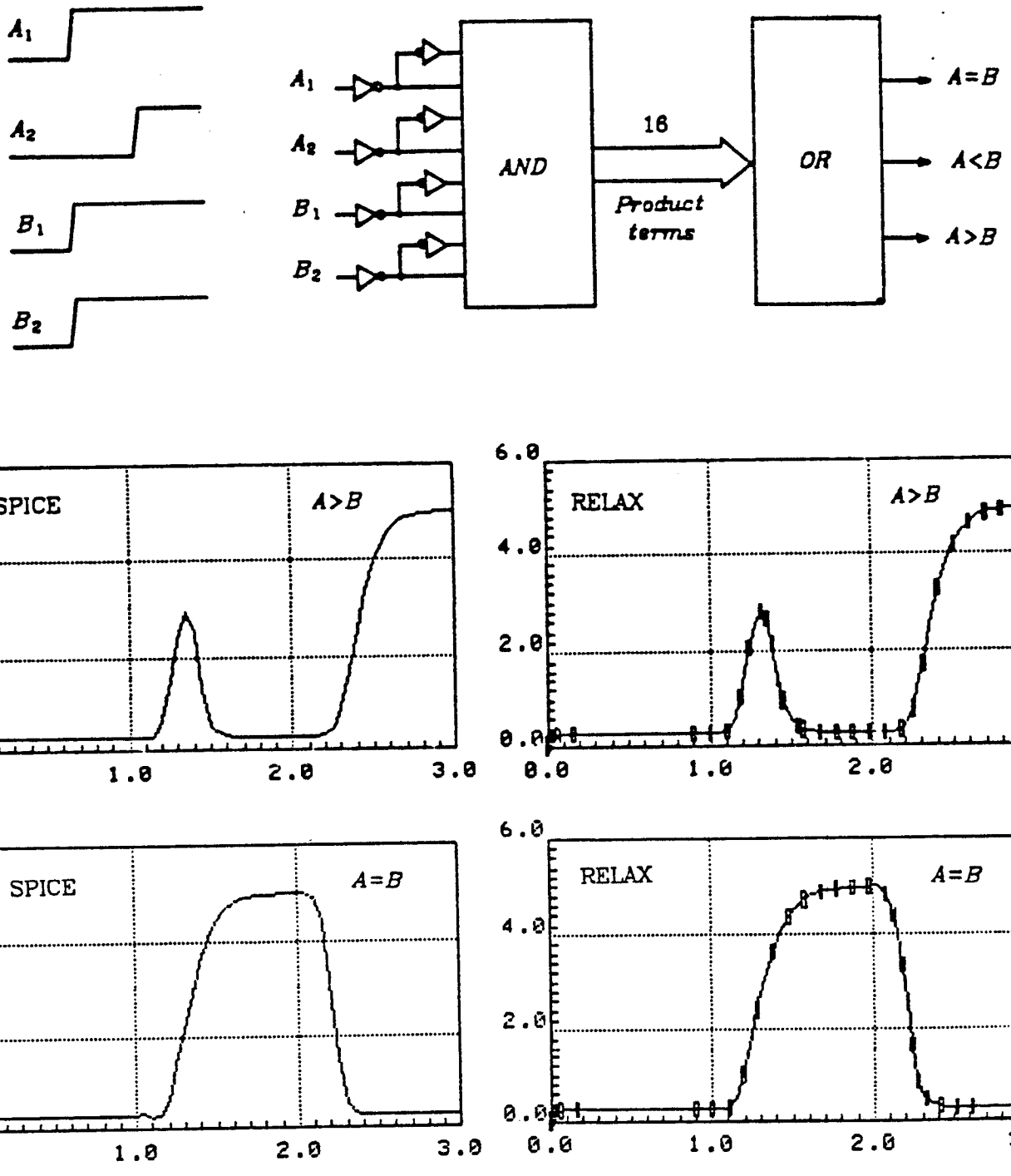



Fig. 9.4

A two-bits magnitude comparator implemented by a NOR-NOR PLA with no minimization of the product terms

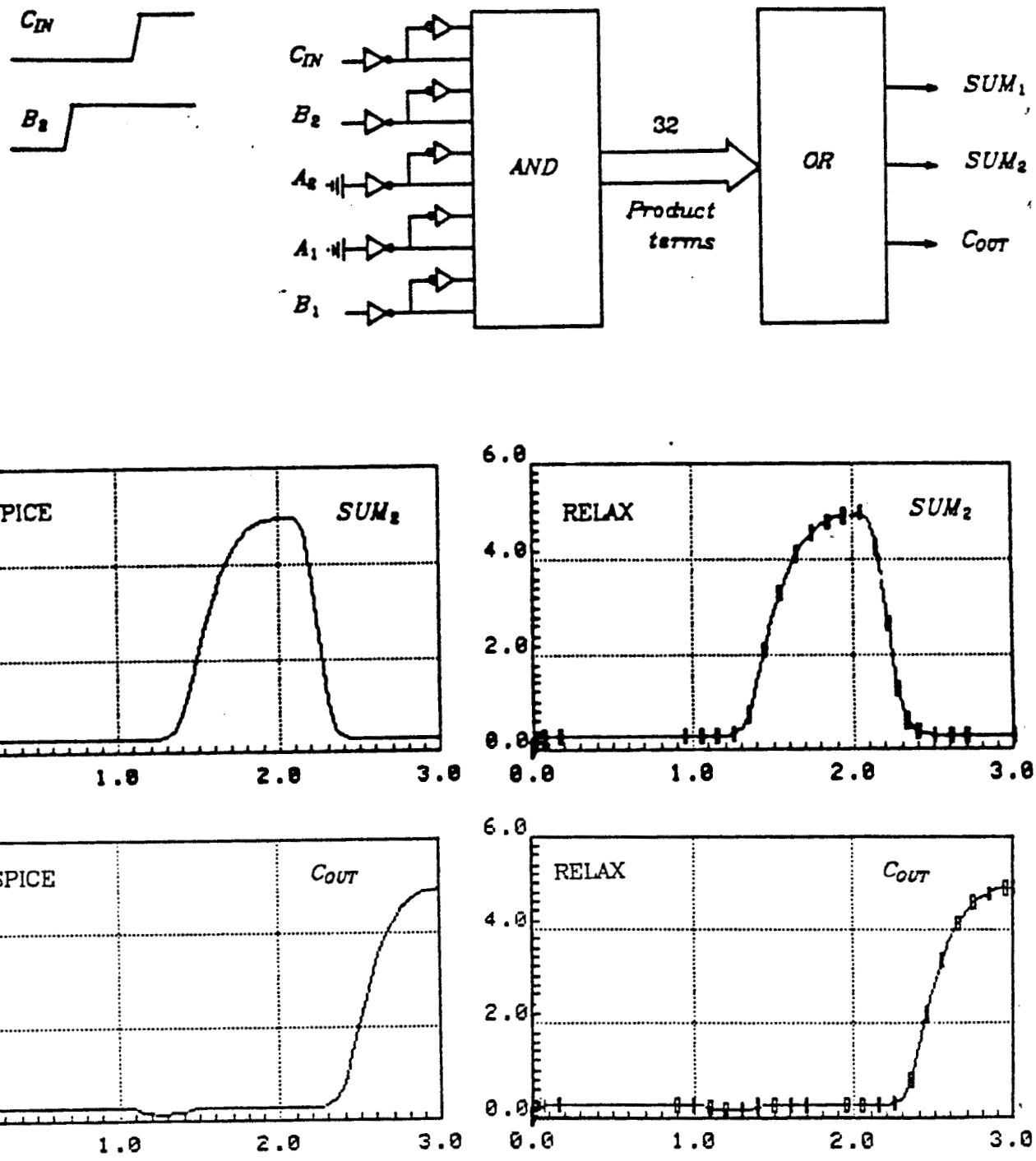



Fig. 9.5

A two-bits full adder implemented by a NOR-NOR PLA with no minimization of the product terms

| Circuit[1] of | Fig. 9.1 | Fig. 9.2 | Fig. 9.3 | Fig. 9.4 | Fig. 9.5 |
|---|---|---|---|---|---|
| # of unknown nodes | 4 | 8 | 16 | 27 | 45 |
| # of MOS devices | 6 | 21 | 42 | 131 | 263 |
| CPU-SPICE (sec) | 21.30 | 121.57 | 211.53 | 818.00 | 1334.80 |
| CPU-RELAX (sec) | 1.08 | 4.38 | 5.85 | 18.42 | 22.30 |
| # of RELAX iterations | 5 | 5 | 7 | 5 | 4 |
| CPU-Ratio ($\frac{\text{SPICE}}{\text{RELAX}}$) | 19.70 | 27.73 | 36.16 | 44.42 | 59.86 |

**Table 9.1**

A comparison of CPU time (in seconds) between RELAX and SPICE2.

[1] With floating capacitors. The ratio of a floating capacitance to a grounded capacitance is approximately 1 to 12.

# Chapter 10

# Conclusion

We have proposed and studied the Waveform Relaxation (WR) method for solving a system of mixed implicit algebraic-differential equations. The key idea behind this method is to apply the relaxation decomposition directly to the given system of equations. As a result, each decomposed subsystem is a dynamical system which can be solved independently by using standard numerical methods. In particular, we have discussed the convergence of the method from a sound theoretical basis. We have also shown that the WR method is guaranteed to converge for a large class of dynamical systems, especially large scale MOS integrated circuits.

We have described the organization and the analysis techniques of a new MOS digital circuit simulator RELAX which implements the WR method. In particular, we have described a few important techniques which account for considerable improvements in the speed of RELAX and make the program suitable for simulating large scale digital integrated circuits. These techniques are:

1) A scheduling technique which improves the speed of convergence of the WR method.

2) The latency and partial waveform convergence techniques which increase the speed of the analysis of each subcircuit.

3) A Look-ahead storage bufferring scheme which enables RELAX to simulate large circuits without using a large amount of primary storage.

Experimental results have indicated that RELAX can exhibit definite improvements over a standard circuit simulator SPICE for simulating MOS digital circuits while maintaining the same accuracy. However, RELAX requires a

number of enhancements before it can become a standard tool for analysing circuits. These enhancements include the implementation of the hierachical input language decription of the circuit, an interface with a logic simulator in order to utilize the information of the logic simulation and the implementation of the table look-up models for MOS devices.

It is clear that the WR method is very suitable for implementing on a computer whose architecture supports parallel and/or pipeline processing since it allows different subcircuits to be analysed concurrently on different processors. More work needs to be done to explore this aspect. We remark here that both GS-WR and GJ-WR methods can be implemented on this type of computer architecture. However, the GJ-WR method may be more suitable since it does not impose any constraint on the order within which the decomposed subcircuits are simulated.

Another area of simulation techniques that needs to be further investigated is the use of an iterative relaxation decomposition at the nonlinear equation level. So far, only one sweep of nonlinear relaxation has been used in the so called "timing simulation" technique which, although may be quite fast, can sometime produce inaccurate results. The use of iteration not only will improve the accuracy of the method but also can detect and inform the user when non-convergence problems arise.

# References

[1] L.N. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," University of California, Berkeley, Electronics Research Laboratory, Memorandum No. ERL-M520, May 1975.

[2] W.T. Weeks, A.J. Jimenez, G.W. Mahoney, D. Mehta, H. Qassemzadeh and T.R. Scott, "Algorithms for ASTAP- A Network Analysis Program," *IEEE Trans. on Circuit Theory*, Vol. CT-20, pp. 628-634, November 1973.

[3] A.R. Newton and D.O. Pederson, "Analysis Time, Accuracy and Memory Requirement Tradeoffs in SPICE2," *IEEE Proc. International Symposium on Circuits and Systems*, pp. 6-9, 1978.

[4] G.D. Hachtel and A.L. Sangiovanni-Vincentelli, "A Survey of Third-Generation Simulation Techniques," *Proceedings of the IEEE*, Vol. 69, No. 10, pp. 1264-1280, October 1981.

[5] A.R. Newton, "The Simulation of Large Scale Integrated Circuits," *IEEE Trans. on Circuits and Systems*, Vol. CAS-26, pp. 741-749, September 1979.

[6] B.R. Chawla, H.K. Gummel and P. Kozak, "MOTIS- An MOS Timing Simulator," *IEEE Trans. on Circuits and Systems*, Vol. CAS-22, pp. 901-910, December 1975.

[7] N.B.G. Rabbat, A.L. Sangiovanni-Vincentelli and H.Y. Hsieh, "A Multilevel Newton Algorithm with Macromodelling and Latency for the Analysis of Large-Scale Nonlinear Circuits in the Time Domain," *IEEE Trans. on Circuits and Systems*, Vol. CAS-26, pp. 733-741, September 1979.

[8] P. Yang, I.N. Hajj and T.N. Trick, "SLATE: A Circuit Simulation Program with Latency Exploitation and Node Tearing," *IEEE Proceedings Int. Conference on Circuits and Computers*, New York, October 1980.

[9] G. Arnout and H. De Man, "The Use of Threshold Functions and Boolean-Controlled Network Elements for Macromodelling of LSI Circuits," *IEEE Journal of Solid-State Circuits*, Vol. SC-13, pp. 326-332, June 1978.

[10] K. Sakallah and S.W. Director, "An Activity-Directed Circuit Simulation Algorithm," *IEEE Proceedings Int. Conference on Circuits and Computers*, New York, pp. 1032-1035, October 1980.

[11] A. Vladimirescu and D.O. Pederson, "A Computer Program for the Simulation of Large Scale Integrated Circuits," *IEEE Proc. International Symposium on Circuits and Systems*, Chicago 1981.

[12] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice Hall, 1971.

[13] A. R. Newton, "The Simulation of Large Scale Integrated Circuits," University of California, Berkeley, Electronics Research Laboratory, Memo. No. ERL-M78/52, July 1978.

[14] J.A. George, "On Block Elimination for Sparse Linear Systems," *SIAM J. Numerical Analysis*, Vol. 11, pp. 585-603, 1974.

[15] A.L. Sangiovanni-Vincentelli, "On the Decomposition of Large Scale Systems of Linear Algebraic Equations," *Proc. of JACC*, Denver, 18-20, June 1979.

[16] J. Sherman and W.J. Morrison, "Adjustment of an Inverse Matrix Corresponding to Changes in the Elements of a Given Column or a Given Row of the Original Matrix," *Amer. Math. Stat.*, Vol. 20, pp. 621, 1949.

[17] R.S. Varga, *Matrix Iterative Analysis*, Prentice Hall, 1962.

[18] G. DeMicheli and A. L. Sangiovanni-Vincentelli, "Numerical Properties of Algorithms for the Timing Analysis of MOS VLSI Circuits," *Proceedings ECCTD'81* The Hague, August 1981.

[19] W. Kahan, Private notes, 1975.

[20] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, 1970.

[21] C. W. Ho, A. E. Ruehli and P. A. Brennan, "The Modified Nodal Approach to Network Analysis," *IEEE Transactions on Circuits and Systems*, Vol. CAS-22, pp. 504-509, June 1975.

[22] G. D. Hachtel, R. K. Brayton and F. G. Gustavson, "The Sparse Tableau Approach to Network Analysis and Design," *IEEE Transactions on Circuit Theory*, Vol. CT-18, pp. 101-113, January 1971.

[23] C.A. Desoer and E.S. Kuh, *Basic Circuit Theory*, McGraw-Hill, 1969.

[24] E.L. Lawler, *Combinatorial Optimization*, Holt, Rinehart and Winston, 1976.

[25] J. K. Hale, *Ordinary Differential Equations*, McGraw-Hill, 1969.

[26] W. Rudin, *Functional Analysis*, McGraw-Hill, 1973.

[27] J.D. Crawford, M.Y. Hsueh, A.R. Newton and D.O. Pederson, *MOTIS-C User's Guide*, Electronics Research Laboratory, University of California, Berkeley, June 1978.

[28] J.R. Bunch and D.J. Rose (editors), *Sparse Matrix Computations*, Academic Press, 1976.

[29] G.R. Case, "The SALOGS Digital Logic Simulator," *Proc. IEEE International Symposium on Circuits and Systems*, New York, pp. 5-10, May 1978.

[30] A. E. Ruehli, A. L. Sangiovanni-Vincentelli and N. B. G. Rabbat, "Time Analysis of Large Scale Circuits Containing One-Way Macromodels," *IEEE Transactions on Circuits and Systems*, Vol. CAS-29, pp. 185-190, March 1982.

# Appendix A

# Proof of Theorems and Lemmas

## Proof of Lemma 4.1

Suppose that (4.10) is false. Then, by interchanging the edges between $L \cap M$ and $L \cap \overline{M}$, we obtain another complete matching whose total weight is larger than the weight of $M$. Hence $M$ is not a maximum weighted complete matching. The converse is trivial. ∎

## Proof of Lemma 4.2

We prove this lemma by contradiction. Suppose that the symbolic number of states of the system is greater than $\sigma - p$. Let $\tilde{Y}$ be the corresponding set of the symbolic state variables and $\tilde{M}$ be any matching which matches at least all the variables in $\tilde{Y}$ to the system equations. Since the time derivative of a symbolic state variable must appear in the system equations and $M$ is a maximum weighted complete matching, we must have that

$$|\tilde{M}| < p$$

i.e., $\tilde{M}$ cannot be a complete matching (otherwise, the weight of $\tilde{M}$ will be larger than the weight of $M$). Hence, given the values of the variables in $\tilde{Y}$ and their time derivatives, there are some system variables that cannot be solved for. Therefore, $\tilde{Y}$ cannot possibly form a state variable of the system and we obtain a contradiction. Hence, the symbolic number of states of the system must equal to $\sigma - p$. It is then obvious that $\{y_v \mid (s,v) \in M \text{ and } w(s,v) = 2\}$ is a set of the symbolic state variables of the system and, by the nondegeneracy assumption 4.1, it is also a set of the state variables. ∎

## Proof of Theorem 5.1

This is a well known theorem in mathematics. We repeat its proof here for the sake of completeness. First, we show that the sequence converges. Here, we have

$$\begin{aligned} \|y^k - y^{k-1}\| &= \|\mathbf{F}(y^{k-1}) - \mathbf{F}(y^{k-2})\| \\ &\leq \gamma\|y^{k-1} - y^{k-2}\| \\ &\leq \gamma^{k-1}\|y^1 - y^0\| \end{aligned}$$

This shows that the sequence is a Cauchy sequence [26]. Since $Y$ is a complete normed space, the sequence converges to a limit $\hat{y} \in Y$.

Next, we show that $\hat{y}$ is a fixed point and is unique. Taking the limit of equation (5.2) as $k \to \infty$ and using the fact that $\mathbf{F}$ is continuous (an obvious consequence of being a contraction map), we obtain

$$\hat{y} = \mathbf{F}(\hat{y})$$

That is, $\hat{y}$ is a fixed point of $\mathbf{F}$. Now suppose that there is another fixed point $\tilde{y} \neq \hat{y}$. Then

$$\|\hat{y} - \tilde{y}\| = \|\mathbf{F}(\hat{y}) - \mathbf{F}(\tilde{y})\| \leq \gamma\|\hat{y} - \tilde{y}\|$$

which is a contradiction since $\gamma < 1$. Therefore, the fixed point of $\mathbf{F}$ is unique.

Finally, we show that the rate of convergence of the fixed point algorithm is linear. Here, we have that

$$\begin{aligned} \|y^k - \hat{y}\| &= \|\mathbf{F}(y^{k-1}) - \mathbf{F}(\hat{y})\| \\ &\leq \gamma\|y^{k-1} - \hat{y}\| \\ &\leq \gamma^k\|y^0 - \hat{y}\| \end{aligned}$$

■

# Proof of Theorem 5.2

a) We have

$$y^{k+1} - \hat{y} = y^{k+1} - F_k(y^k) + F_k(y^k) - F_k(\hat{y}) + F_k(\hat{y}) - F(\hat{y})$$

Hence

$$\begin{aligned} \|y^{k+1} - \hat{y}\| &\le \varepsilon_k + \gamma_k \|y^k - \hat{y}\| + \delta_k \\ &= \gamma_k \|y^k - \hat{y}\| + \alpha_k \\ &\le \beta_{k0}\gamma_0 \|y^0 - \hat{y}\| + \sum_{j=0}^{k} \beta_{kj}\alpha_j \end{aligned} \tag{A1.1}$$

b) We have

$$\begin{aligned} y^{k+1} - \hat{y} = {} & y^{k+1} - F_k(y^k) + F_k(y^k) - F_k(y^{k+1}) \\ & + F_k(y^{k+1}) - F_k(\hat{y}) + F_k(\hat{y}) - F(\hat{y}) \end{aligned}$$

Hence

$$\|y^{k+1} - \hat{y}\| \le \varepsilon_k + \gamma_k \|y^{k+1} - y^k\| + \gamma_k \|y^{k+1} - \hat{y}\| + \delta_k$$

Since $1 - \gamma_k > 0$, the above inequality implies that

$$\|y^{k+1} - \hat{y}\| \le \frac{\gamma_k}{1 - \gamma_k} \|y^{k+1} - \hat{y}\| + \frac{\alpha_k}{1 - \gamma_k} \tag{A1.2}$$

c) We shall show that, given any $\delta > 0$, there exists an integer $k_\delta$ such that

$$\|y^{k+1} - \hat{y}\| \le \delta \qquad \text{for all } k \ge k_\delta.$$

From (A1.1), we have, for any $k \ge k_1$, that

$$\|y^{k+1} - \hat{y}\| \le \beta_{k0}\gamma_0 \|y^0 - \hat{y}\| + \sum_{j=0}^{k_1} \beta_{kj}\alpha_j + \sum_{j=k_1+1}^{k} \beta_{kj}\alpha_j \tag{A1.3}$$

Since $\lim_{k\to\infty} \alpha_k = 0$ and $\lim_{k\to\infty} \beta_{k0} = 0$, we can choose $k_1$ such that

$$\alpha_k \le \frac{\delta}{3c} \quad \text{and} \quad \beta_{k0}\gamma_0 \|y^0 - \hat{y}\| \le \frac{\delta}{3} \quad \text{for all } k \ge k_1$$

Hence (A1.3) becomes

$$\|y^{k+1} - \hat{y}\| \le \frac{\delta}{3} + \sum_{j=0}^{k_1} \beta_{kk_1}\beta_{k_1 j}\alpha_j + \frac{\delta}{3c}\sum_{j=k_1+1}^{k} \beta_{kj}$$

$$\le \frac{\delta}{3} + \beta_{kk_1}\sum_{j=0}^{k_1} \beta_{k_1 j}\alpha_j + \frac{\delta}{3c}\sum_{j=0}^{k} \beta_{kj} \qquad \text{for all } k \ge k_1 \qquad \text{(A1.4)}$$

Since $\lim_{k\to\infty} \sum_{j=0}^{k} \beta_{kj} = c$, therefore (A1.4) becomes

$$\|y^{k+1} - \hat{y}\| \le \frac{2\delta}{3} + \beta_{kk_1}\sum_{j=0}^{k_1} \beta_{k_1 j}\alpha_j \qquad \text{for all } k \ge k_1 \qquad \text{(A1.5)}$$

Since $\lim_{k\to\infty} \beta_{kk_1} = 0$, we can choose $k_2 \ge k_1$ such that

$$\beta_{kk_1}\sum_{j=0}^{k_1} \beta_{k_1 j}\alpha_j \le \frac{\delta}{3} \qquad \text{for all } k \ge k_2$$

Therefore, (A1.5) becomes

$$\|y^{k+1} - \hat{y}\| \le \delta \qquad \text{for all } k \ge k_2$$

and the proof is then complete. ∎

**Proof of Corollary 5.2**

Here we have

$$\beta_{kj} \le \tilde{\gamma}^{k-j} \qquad \text{for any } j \le k$$

Thus

$$\sum_{j=0}^{k} \beta_{kj} \le \sum_{j=0}^{k} \tilde{\gamma}^{k-j} = \frac{1-\tilde{\gamma}^{k+1}}{1-\tilde{\gamma}}$$

$$\le \frac{1}{1-\tilde{\gamma}} \qquad \text{for any } k.$$

Therefore, $\beta_{kj}$ satisfies the condition (c) of Theorem 5.2. Hence

$$\lim_{k\to\infty} y^k = \hat{y} \qquad \text{if} \qquad \lim_{k\to\infty} \alpha_k = 0$$

∎

The following fact is useful for the proof of Theorem 5.3.

**Lemma A.1** Let $\|\cdot\|_a$, $\|\cdot\|_b$ be norms in $\mathbf{R}^n$, $\mathbf{R}^{n+l}$ respectively. Then there exists a constant $\mu$ such that

$$\|x\|_a \leq \mu \left\| \begin{matrix} x \\ z \end{matrix} \right\|_b \qquad \text{for all } x \in \mathbf{R}^n \text{ and } z \in \mathbf{R}^l$$

**Proof** We use the fact that all norms in a finite dimensional space are equivalent (see [20] page 39). That is, if $\|\cdot\|_a$ and $\|\cdot\|_{\tilde{a}}$ are norms in $\mathbf{R}^n$, there exist constants $\mu_1$ and $\mu_2$ such that, for any $x \in \mathbf{R}^n$,

$$\|x\|_a \leq \mu_1 \|x\|_{\tilde{a}} \qquad \text{and} \qquad \|x\|_{\tilde{a}} \leq \mu_2 \|x\|_a \tag{A2.1}$$

Define

$$\|x\|_{\tilde{a}} \triangleq \left\| \begin{matrix} x \\ 0 \end{matrix} \right\|_b \qquad \text{and} \qquad \left\| \begin{matrix} x \\ z \end{matrix} \right\|_{\tilde{b}} \triangleq \left\| \begin{matrix} x \\ 0 \end{matrix} \right\|_b + \left\| \begin{matrix} 0 \\ z \end{matrix} \right\|_b \tag{A2.2}$$

Then from (A2.1), there exist $\mu_1$, $\tilde{\mu}_2$ such that

$$\|x\|_a \leq \mu_1 \|x\|_{\tilde{a}} \qquad \text{and} \qquad \left\| \begin{matrix} x \\ z \end{matrix} \right\|_{\tilde{b}} \leq \tilde{\mu}_2 \left\| \begin{matrix} x \\ z \end{matrix} \right\|_b \tag{A2.3}$$

(A2.2) and (A2.3) imply that

$$\begin{aligned} \|x\|_a &\leq \mu_1 \|x\|_{\tilde{a}} \\ &\leq \mu_1 \left\| \begin{matrix} x \\ z \end{matrix} \right\|_{\tilde{b}} \\ &\leq \mu_1 \tilde{\mu}_2 \left\| \begin{matrix} x \\ z \end{matrix} \right\|_b \end{aligned}$$

Therefore, the proof is then completed. ∎

# Proof of Theorem 5.3

Define

$$\mathbf{D} \triangleq \{t \in [0,T] \mid t \text{ is a discontinuity point of either } u(\cdot),\ \dot{x}^0(\cdot) \text{ or } z^0(\cdot)\}$$

$$\mathbf{X}\times\mathbf{Z} \triangleq \{(x(\cdot), z(\cdot)) : [0,T]\to\mathbf{R}^n\times\mathbf{R}^l \mid \dot{x}(\cdot) \text{ and } z(\cdot) \text{ are piecewise continuous with possible discontinuity points in } \mathbf{D}\}$$

$$\mathbf{F} : \mathbf{X}\times\mathbf{Z} \to \mathbf{X}\times\mathbf{Z} \text{ such that } \begin{bmatrix}\tilde{x}(\cdot)\\ \tilde{z}(\cdot)\end{bmatrix} = \mathbf{F}\begin{bmatrix}x(\cdot)\\ z(\cdot)\end{bmatrix} \text{ satisfies}$$

$$\dot{\tilde{x}} = f(\tilde{x}, x, \dot{x}, z, u)\ ; \qquad \tilde{x}(0) = x_0 \tag{A3.1a}$$

$$\tilde{z} = g(\tilde{x}, x, \dot{x}, z, u) \tag{A3.1b}$$

Since $f$ is Lipschitz continuous with respect to $\tilde{x}$, by the Picard-Lindelöf Theorem on the existence and uniqueness of the solutions of ordinary differential equations (see [25] page 18), the above equations have a unique solution $(\tilde{x}(\cdot), \tilde{z}(\cdot))$ which belongs to $\mathbf{X}\times\mathbf{Z}$. Therefore $\mathbf{F}$ is well defined. From these definitions the canonical WR algorithm described by (5.12) can be written as

$$\begin{bmatrix}x^k(\cdot)\\ z^k(\cdot)\end{bmatrix} = \mathbf{F}\begin{bmatrix}x^{k-1}(\cdot)\\ z^{k-1}(\cdot)\end{bmatrix} \tag{A3.2}$$

Let $\hat{\gamma} = \max(\gamma, 0.1)$, $\hat{\lambda}_1 = \lambda_1$, $\hat{\lambda}_2 = \lambda_2$ and $\eta$ be a positive constant whose value will be chosen later. Define norms in $\mathbf{R}^n\times\mathbf{R}^l\times\mathbf{R}^n$ and $\mathbf{X}\times\mathbf{Z}$ as follows:

$$\left\|\begin{matrix}\dot{x}(t)\\ z(t)\\ x(t)\end{matrix}\right\| \triangleq \left\|\begin{matrix}\dot{x}(t)\\ z(t)\end{matrix}\right\| + \frac{\hat{\lambda}_2}{\hat{\gamma}}\|x(t)\| \tag{A3.3}$$

$$\left\|\begin{matrix}x(\cdot)\\ z(\cdot)\end{matrix}\right\| \triangleq \max_{t\in[0,T]} e^{-\eta t}\left\|\begin{matrix}\dot{x}(t)\\ z(t)\\ x(t)\end{matrix}\right\| \tag{A3.4}$$

where the norms in $\mathbf{R}^n\times\mathbf{R}^l$ and $\mathbf{R}^n$ are as given in Theorem 5.3. Since

$$\left\|\begin{matrix} x(\cdot) \\ z(\cdot) \end{matrix}\right\| \geq e^{-\eta T} \max_{t \in [0,T]} \left\|\begin{matrix} \dot{x}(t) \\ z(t) \\ x(t) \end{matrix}\right\|$$

Therefore, it can be shown that the space $(X \times Z, \|\cdot\|)$ is closed (hence it is a Banach space). Next we shall show that $F$ is contractive in $(X \times Z, \|\cdot\|)$. Let

$$\begin{bmatrix} \tilde{x}^1(\cdot) \\ \tilde{z}^1(\cdot) \end{bmatrix} = F \begin{bmatrix} x^1(\cdot) \\ z^1(\cdot) \end{bmatrix} \qquad \text{and} \qquad \begin{bmatrix} \tilde{x}^2(\cdot) \\ \tilde{z}^2(\cdot) \end{bmatrix} = F \begin{bmatrix} x^2(\cdot) \\ z^2(\cdot) \end{bmatrix}$$

Then, from (A3.1) and the condition (b) of Theorem 5.3, we have

$$\begin{aligned} \left\|\begin{matrix} \dot{\tilde{x}}^1(t) - \dot{\tilde{x}}^2(t) \\ \tilde{z}^1(t) - \tilde{z}^2(t) \end{matrix}\right\| &= \left\|\begin{matrix} f(\tilde{x}^1, x^1, \dot{x}^1, z^1, u) - f(\tilde{x}^2, x^2, \dot{x}^2, z^2, u) \\ g(\tilde{x}^1, x^1, \dot{x}^1, z^1, u) - g(\tilde{x}^2, x^2, \dot{x}^2, z^2, u) \end{matrix}\right\| \\ &\leq \hat{\lambda}_1 \|\tilde{x}^1(t) - \tilde{x}^2(t)\| + \hat{\lambda}_2 \|x^1(t) - x^2(t)\| + \gamma \left\|\begin{matrix} \dot{x}^1(t) - \dot{x}^2(t) \\ z^1(t) - z^2(t) \end{matrix}\right\| \end{aligned}$$

After some algebraic manipulations using (A3.3), $\hat{\gamma}$ and the above inequality, we have

$$\left\|\begin{matrix} \dot{\tilde{x}}^1(t) - \dot{\tilde{x}}^2(t) \\ \tilde{z}^1(t) - \tilde{z}^2(t) \\ \tilde{x}^1(t) - \tilde{x}^2(t) \end{matrix}\right\| \leq \hat{\gamma} \left\|\begin{matrix} \dot{x}^1(t) - \dot{x}^2(t) \\ z^1(t) - z^2(t) \\ x^1(t) - x^2(t) \end{matrix}\right\| + (\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}}) \|\tilde{x}^1(t) - \tilde{x}^2(t)\| \qquad \text{(A3.5)}$$

From Lemma A.1, there exists a constant $\mu$ such that

$$\|\dot{\tilde{x}}^1(t) - \dot{\tilde{x}}^2(t)\| \leq \mu \left\|\begin{matrix} \dot{\tilde{x}}^1(t) - \dot{\tilde{x}}^2(t) \\ \tilde{z}^1(t) - \tilde{z}^2(t) \\ \tilde{x}^1(t) - \tilde{x}^2(t) \end{matrix}\right\| \qquad \text{(A3.6)}$$

From (A3.5) and (A3.6), letting $\mu_1 = \mu\hat{\gamma}$ and $\mu_2 = \mu(\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}})$, we have

$$\|\dot{\tilde{x}}^1(t) - \dot{\tilde{x}}^2(t)\| \leq \mu_1 \left\|\begin{matrix} \dot{x}^1(t) - \dot{x}^2(t) \\ z^1(t) - z^2(t) \\ x^1(t) - x^2(t) \end{matrix}\right\| + \mu_2 \|\tilde{x}^1(t) - \tilde{x}^2(t)\| \qquad \text{(A3.7)}$$

Applying the fundamental results in differential inequalities to (A3.7) (see [25] corollary 6.2, pages 30-32), using the fact that $\tilde{x}^1(0) - \tilde{x}^2(0) = 0$, we have

$$\|\tilde{x}^1(t) - \tilde{x}^2(t)\| \leq \mu_1 e^{\mu_2 t} \int_0^t e^{-\mu_2 \tau} \left\| \begin{matrix} \dot{x}^1(\tau) - \dot{x}^2(\tau) \\ z^1(\tau) - z^2(\tau) \\ x^1(\tau) - x^2(\tau) \end{matrix} \right\| d\tau$$

$$= \mu_1 e^{\mu_2 t} \int_0^t e^{(\eta-\mu_2)\tau} e^{-\eta\tau} \left\| \begin{matrix} \dot{x}^1(\tau) - \dot{x}^2(\tau) \\ z^1(\tau) - z^2(\tau) \\ x^1(\tau) - x^2(\tau) \end{matrix} \right\| d\tau \qquad \text{(A3.8)}$$

From (A3.4) we have

$$e^{-\eta\tau} \left\| \begin{matrix} \dot{x}^1(\tau) - \dot{x}^2(\tau) \\ z^1(\tau) - z^2(\tau) \\ x^1(\tau) - x^2(\tau) \end{matrix} \right\| \leq \left\| \begin{matrix} x^1(\cdot) - x^2(\cdot) \\ z^1(\cdot) - z^2(\cdot) \end{matrix} \right\| \qquad \text{for all } \tau \in [0, T]$$

Substituting this inequality in (A3.8), we have

$$\|\tilde{x}^1(t) - \tilde{x}^2(t)\| \leq \mu_1 e^{\mu_2 t} \left\| \begin{matrix} x^1(\cdot) - x^2(\cdot) \\ z^1(\cdot) - z^2(\cdot) \end{matrix} \right\| \int_0^t e^{(\eta-\mu_2)\tau} d\tau$$

$$= \frac{\mu_1 e^{\mu_2 t}}{\eta - \mu_2} \left\| \begin{matrix} x^1(\cdot) - x^2(\cdot) \\ z^1(\cdot) - z^2(\cdot) \end{matrix} \right\| (e^{(\eta-\mu_2)t} - 1)$$

$$\leq \frac{\mu_1 e^{\eta t}}{\eta - \mu_2} \left\| \begin{matrix} x^1(\cdot) - x^2(\cdot) \\ z^1(\cdot) - z^2(\cdot) \end{matrix} \right\| \qquad \text{assuming } \eta - \mu_2 > 0$$

Substituting this inequality in (A3.5) and multiplying by $e^{-\eta t}$, we have

$$e^{-\eta t} \left\| \begin{matrix} \dot{\tilde{x}}^1(t) - \dot{\tilde{x}}^2(t) \\ \tilde{z}^1(t) - \tilde{z}^2(t) \\ \tilde{x}^1(t) - \tilde{x}^2(t) \end{matrix} \right\| \leq \hat{\gamma} e^{-\eta t} \left\| \begin{matrix} \dot{x}^1(t) - \dot{x}^2(t) \\ z^1(t) - z^2(t) \\ x^1(t) - x^2(t) \end{matrix} \right\| + \frac{\mu_1(\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}})}{\eta - \mu_2} \left\| \begin{matrix} x^1(\cdot) - x^2(\cdot) \\ z^1(\cdot) - z^2(\cdot) \end{matrix} \right\|$$

Hence, using (A3.4), the above inequality implies that

$$\left\| \begin{matrix} \tilde{x}^1(\cdot) - \tilde{x}^2(\cdot) \\ \tilde{z}^1(\cdot) - \tilde{z}^2(\cdot) \end{matrix} \right\| \leq \hat{\gamma} \left\| \begin{matrix} x^1(\cdot) - x^2(\cdot) \\ z^1(\cdot) - z^2(\cdot) \end{matrix} \right\| + \frac{\mu_1(\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}})}{\eta - \mu_2} \left\| \begin{matrix} x^1(\cdot) - x^2(\cdot) \\ z^1(\cdot) - z^2(\cdot) \end{matrix} \right\|$$

That is

$$\left\| \mathbf{F} \begin{bmatrix} x^1(\cdot) \\ z^1(\cdot) \end{bmatrix} - \mathbf{F} \begin{bmatrix} x^2(\cdot) \\ z^2(\cdot) \end{bmatrix} \right\| \leq \left[ \hat{\gamma} + \frac{\mu_1(\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}})}{\eta - \mu_2} \right] \left\| \begin{matrix} x^1(\cdot) - x^2(\cdot) \\ z^1(\cdot) - z^2(\cdot) \end{matrix} \right\| \qquad \text{(A3.9)}$$

Since $0.1 \leq \hat{\gamma} < 1$, we can choose $\eta = \hat{\eta}$ such that

$$\hat{\gamma} + \frac{\mu_1(\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}})}{\hat{\eta} - \mu_2} = \frac{1+\hat{\gamma}}{2} < 1 \tag{A3.10}$$

which gives

$$\hat{\eta} = \mu(\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}})\frac{1+\hat{\gamma}}{1-\hat{\gamma}} = \mu_2\frac{1+\hat{\gamma}}{1-\hat{\gamma}} \tag{A3.11}$$

Hence, (A3.9) becomes

$$\left\| \mathbf{F}\begin{bmatrix} x^1(\cdot) \\ z^1(\cdot) \end{bmatrix} - \mathbf{F}\begin{bmatrix} x^2(\cdot) \\ z^2(\cdot) \end{bmatrix} \right\| \leq \frac{1+\hat{\gamma}}{2} \left\| \begin{matrix} x^1(\cdot) - x^2(\cdot) \\ z^1(\cdot) - z^2(\cdot) \end{matrix} \right\|$$

Therefore $\mathbf{F}$ is contractive. Hence, by the Contraction Mapping Theorem 5.1, it has a unique fixed point in $\mathbf{X}\times\mathbf{Z}$ satisfying

$$\begin{bmatrix} \hat{x}(\cdot) \\ \hat{z}(\cdot) \end{bmatrix} = \mathbf{F}\begin{bmatrix} \hat{x}(\cdot) \\ \hat{z}(\cdot) \end{bmatrix}$$

That is

$$\begin{aligned} \dot{\hat{x}} &= f(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) \; ; \qquad \hat{x}(0) = x_0 \\ \hat{z} &= g(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) \end{aligned}$$

Furthermore, for any given initial guess $(x^0(\cdot), z^0(\cdot)) \in \mathbf{X}\times\mathbf{Z}$, the sequence $\{(x^k(\cdot), z^k(\cdot))\}_{k=1}^{\infty}$ generated by the fixed point algorithm (A3.2) converges uniformly to $(\hat{x}(\cdot), \hat{z}(\cdot)) \in \mathbf{X}\times\mathbf{Z}$. Moreover, since we can choose $(x^0(\cdot), z^0(\cdot)) \in \mathbf{X}\times\mathbf{Z}$ such that $x^0(t) = x_0$ and $z^0(t) = 0$ for all $t \in [0, T]$, we conclude that the discontinuity points of $(\hat{x}(\cdot), \hat{z}(\cdot))$ belong to the set of discontinuity points of $u(\cdot)$ only, i.e., they do not depend on $\dot{x}^0(\cdot)$ and $z^0(\cdot)$. Hence the proof is then completed. ∎

## Proof of Theorem 5.4

Here we use the same notations as in Theorem 5.3 and its proof. Let $\{x^k(t), z^k(t) ; t \in [0,T]\}_{k=1}^{\infty}$ be the sequence generated by the following canonical WR algorithm.

$$\dot{x}^k = f(x^k, x^{k-1}, \dot{x}^{k-1}, z^{k-1}, u) ; \quad x^k(0) = x_0^k \qquad \text{(A4.1a)}$$

$$z^k = g(x^k, x^{k-1}, \dot{x}^{k-1}, z^{k-1}, u) \qquad \text{(A4.1b)}$$

Let

$$\begin{bmatrix} \tilde{x}^k(\cdot) \\ \tilde{z}^k(\cdot) \end{bmatrix} = \mathbf{F}\begin{bmatrix} x^{k-1}(\cdot) \\ z^{k-1}(\cdot) \end{bmatrix}$$

That is

$$\dot{\tilde{x}}^k = f(\tilde{x}^k, x^{k-1}, \dot{x}^{k-1}, z^{k-1}, u) ; \quad \tilde{x}^k(0) = x_0 \qquad \text{(A4.2a)}$$

$$\tilde{z}^k = g(\tilde{x}^k, x^{k-1}, \dot{x}^{k-1}, z^{k-1}, u) \qquad \text{(A4.2b)}$$

Subtracting (A4.1) from (A4.2), we obtain

$$\left\| \begin{matrix} \dot{\tilde{x}}^k(t) - \dot{x}^k(t) \\ \tilde{z}^k(t) - z^k(t) \end{matrix} \right\| \leq \hat{\lambda}_1 \|\tilde{x}^k(t) - x^k(t)\|$$

From (A3.3) and the above inequality, we have

$$\left\| \begin{matrix} \dot{\tilde{x}}^k(t) - \dot{x}^k(t) \\ \tilde{z}^k(t) - z^k(t) \\ \tilde{x}^k(t) - x^k(t) \end{matrix} \right\| \leq (\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}}) \|\tilde{x}^k(t) - x^k(t)\| \qquad \text{(A4.3)}$$

Let $\mu_2 = \mu(\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}})$. Then from Lemma A.1 and the above inequality, we have

$$\|\dot{\tilde{x}}^k(t) - \dot{x}^k(t)\| \leq \mu_2 \|\tilde{x}^k(t) - x^k(t)\| \qquad \text{(A4.4)}$$

Applying the fundamental results in differential inequalities to (A4.4), using the fact that $\tilde{x}^k(0) - x^k(0) = x_0 - x_0^k$, we have

$$\|\tilde{x}^k(t) - x^k(t)\| \leq e^{\mu_2 t} \|x_0 - x_0^k\| \qquad \text{(A4.5)}$$

Substituting this inequality in (A4.3), we have

$$\left\| \begin{matrix} \dot{\tilde{x}}^k(t) - \dot{x}^k(t) \\ \tilde{z}^k(t) - z^k(t) \\ \tilde{x}^k(t) - x^k(t) \end{matrix} \right\| \leq (\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}}) e^{\mu_2 t} \|x_0 - x_0^k\| \tag{A4.6}$$

From (A3.4), (A4.6) and the fact that $m_2 - \eta < 0$, we have

$$\begin{aligned} \left\| \begin{matrix} \tilde{x}^k(\cdot) - x^k(\cdot) \\ \tilde{z}^k(\cdot) - z^k(\cdot) \end{matrix} \right\| &\leq (\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}}) \|x_0 - x_0^k\| \max_{t \in [0,T]} e^{(\mu_2 - \eta)t} \\ &\leq (\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}}) \|x_0 - x_0^k\| \end{aligned} \tag{A4.7}$$

Hence

$$\begin{aligned} \varepsilon_{k-1} &\triangleq \left\| \mathbf{F} \begin{bmatrix} x^{k-1}(\cdot) \\ z^{k-1}(\cdot) \end{bmatrix} - \begin{bmatrix} x^k(\cdot) \\ z^k(\cdot) \end{bmatrix} \right\| \\ &\leq (\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}}) \|x_0 - x_0^k\| \end{aligned}$$

Since $\lim_{k \to \infty} x_0^k = x_0$, therefore

$$\lim_{k \to \infty} \varepsilon_k = 0$$

Hence, applying Corollary 5.2 with $\mathbf{F}_k = \mathbf{F}$ (i.e., $\delta_k = 0$), we conclude that the sequence of $(x^k(\cdot), z^k(\cdot))$ generated by (A4.1) converges uniformly to the fixed point of $\mathbf{F}$ which is the solution of (5.13). ∎

## Proof of Theorem 5.5

We use the same notations as in the proof of Theorem 5.3 with the following changes.

$$\begin{aligned}
\hat{\gamma} &= \max(\gamma, 0.1, \gamma_M) < 1 \\
\hat{\lambda}_1 &= \max(\lambda_1, \lambda_{1M}) \\
\hat{\lambda}_2 &= \max(\lambda_1, \lambda_{2M}) \\
\hat{\eta} &= \mu\left(\hat{\lambda}_1 + \frac{\hat{\lambda}_2}{0.1}\right)\frac{1+\hat{\gamma}}{1-\hat{\gamma}}
\end{aligned}$$

Now, define $\mathbf{F}_k : \mathbf{X}\times\mathbf{Z}\to\mathbf{X}\times\mathbf{Z}$; $k = 1,2,\ldots,\infty$ such that $\begin{bmatrix} x^k(\cdot) \\ z^k(\cdot) \end{bmatrix} = \mathbf{F}_{k-1}\begin{bmatrix} x^{k-1}(\cdot) \\ z^{k-1}(\cdot) \end{bmatrix}$ satisfies (5.15). Then, by using the proof similar to the proof of Theorem 5.3, we can show that $\mathbf{F}$, $\mathbf{F}_k$, $k = 1,2,\ldots,\infty$ are all contraction maps with contraction constants less than $\tilde{\gamma}$ where

$$\tilde{\gamma} \le \frac{1+\hat{\gamma}}{2} < 1$$

Let

$$\delta_k \triangleq \left\| \mathbf{F}_k\begin{bmatrix} \hat{x}(\cdot) \\ \hat{z}(\cdot) \end{bmatrix} - \mathbf{F}\begin{bmatrix} \hat{x}(\cdot) \\ \hat{z}(\cdot) \end{bmatrix} \right\| \qquad \text{(A5.1)}$$

Then, by applying Corollary 5.2 with $\alpha_k = 0$, the proof of this Theorem is completed when we can show that $\lim_{k\to\infty}\delta_k = 0$. Let

$$\begin{bmatrix} \hat{x}^k(\cdot) \\ \hat{z}^k(\cdot) \end{bmatrix} = \mathbf{F}_k\begin{bmatrix} \hat{x}(\cdot) \\ \hat{z}(\cdot) \end{bmatrix}$$

That is

$$\dot{\hat{x}}^k = f^{k+1}(\hat{x}^k, \hat{x}, \dot{\hat{x}}, \hat{z}, u) \; ; \quad \hat{x}^k(0) = x_0 \qquad \text{(A5.2a)}$$

$$\hat{z}^k = g^{k+1}(\hat{x}^k, \hat{x}, \dot{\hat{x}}, \hat{z}, u) \qquad \text{(A5.2b)}$$

Subtracting (5.13) from (A5.2), we have

$$\begin{bmatrix} \dot{\hat{x}}^k - \dot{\hat{x}} \\ \hat{z}^k - \hat{z} \end{bmatrix} = \begin{bmatrix} f^{k+1}(\hat{x}^k, \hat{x}, \dot{\hat{x}}, \hat{z}, u) - f^{k+1}(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) \\ g^{k+1}(\hat{x}^k, \hat{x}, \dot{\hat{x}}, \hat{z}, u) - g^{k+1}(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) \end{bmatrix}$$

$$+ \begin{bmatrix} f^{k+1}(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) - f(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) \\ g^{k+1}(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) - g(\hat{x}, \hat{x}, \dot{\hat{x}}, \hat{z}, u) \end{bmatrix}$$

Hence

$$\left\| \begin{matrix} \dot{\hat{x}}^k(t) - \dot{\hat{x}}(t) \\ \hat{z}^k(t) - \hat{z}(t) \end{matrix} \right\| \leq \hat{\lambda}_1 \|\hat{x}^k(t) - \hat{x}(t)\| + \xi_{k+1}(t)$$

By following similar steps in the proof of Theorem 5.3, we can show that

$$\left\| \begin{matrix} \dot{\hat{x}}^k(t) - \dot{\hat{x}}(t) \\ \hat{z}^k(t) - \hat{z}(t) \\ \hat{x}^k(t) - \hat{x}(t) \end{matrix} \right\| \leq \left( \hat{\lambda}_1 + \frac{\hat{\lambda}_2}{\hat{\gamma}} \right) \|\hat{x}^k(t) - \hat{x}(t)\| + \xi_{k+1}(t) \qquad \text{(A5.3)}$$

$$\|\dot{\hat{x}}^k(t) - \dot{\hat{x}}(t)\| \leq \mu_2 \|\hat{x}^k(t) - \hat{x}(t)\| + \mu \xi_{k+1}(t) \qquad \text{(A5.4)}$$

Applying the fundamental results in differential inequalities to (A5.4), using the fact that $\hat{x}^k(0) - \hat{x}(0) = 0$, we have

$$\|\hat{x}^k(t) - \hat{x}(t)\| \leq \mu \int_0^t e^{\mu_2(t-\tau)} \xi_{k+1}(\tau) d\tau \qquad \text{(A5.5)}$$

Substituing (A5.5) into (A5.3), we have

$$\left\| \begin{matrix} \dot{\hat{x}}^k(t) - \dot{\hat{x}}(t) \\ \hat{z}^k(t) - \hat{z}(t) \\ \hat{x}^k(t) - \hat{x}(t) \end{matrix} \right\| \leq \mu_2 \int_0^t e^{\mu_2(t-\tau)} \xi_{k+1}(\tau) d\tau + \xi_{k+1}(t) \qquad \text{(A5.6)}$$

Since $\lim_{k \to \infty} \xi_k(t) = 0$ for any $t \in [0, T]$, (A5.6), (A5.1) and (A3.4) imply that

$$\lim_{k \to \infty} \delta_k = 0$$

Hence, the proof is completed. ∎

**Proof of Corollary 5.5**

The proof of this corollary follows immediately from the proofs of Theorem 5.4, Theorem 5.5 and Corollary 5.2. ∎

# Proof of Lemma 5.1

We shall prove this lemma by specifying the steps required to transform the WR algorithm into a canonical form. These steps have been described informally in the examples of Chapter 5. In this proof we shall describe these steps in terms of their graph representations. In particular, the process of differentiating the $i$-th equation $\psi$ times will be represented as the process of adding $\psi$ to the weight of all edges incident to the $i$-th node of $S$ where $G = (S,V,B)$ is the weighted bipartite graph associated with the system equations. We shall refer to this procedure as $ADDW(\psi,i,G)$.

**Definition A.1** Let $G = (S,V,B)$ be a weighted bipartite graph associated with a given system of equations as defined in Definition 4.4, $i \in S$ and $\psi$ be an integer. The following procedure is defined as $ADDW(\psi,i,G)$.

For each edge $(k,j) \in B$

If $k = i$, then $\quad w(k,j) = w(k,j) + \psi$

Otherwise, $\quad w(k,j) = w(k,j)$. ∎

**Definition A.2** Let $G = (S,V,B)$ be a weighted bipartite graph with $|S| = |V|$ and $M$ be a matching of $G$. A node $j \in V$ is said to be *dominant* with respect to $M$ if there exists an edge $(i,j) \in M$ and

$$w(i,j) \geq w(k,j) \qquad \text{for all} \quad (k,j) \in B$$

i.e., there is an edge in $M$ that is incident to $j \in V$ and its weight is larger than or equal to the weight of any other edge incident to $j$.

$M$ is said to be a *dominant* matching if all nodes in $V$ that are matched by $M$ are dominant with respect to $M$. ∎

**Facts**

a) If $M$ is a maximum weighted complete matching of $G$, then *ADDW* will not destroy its maximum weight property. That is, $M$ is still a maximum weighted complete matching of the graph after applying *ADDW*.

b) If $M$ is both complete and dominant, then $M$ is also a maximum weighted complete matching.[1] ∎

For the sake of simplicity, we shall consider the case in which the decomposition is pointwise. The extension to the general case is straight forward and hence will be omitted. Without loss of generality,[2] we assume that the maximum weighted complete matching associated with the consistent AP process of the WR algorithm is

$$M = \{(1,1), (2,2), \ldots, (p,p)\} \tag{A6.1}$$

and that

$$w(i,i) = \begin{cases} 2 & 1 \le i \le \sigma - p \\ 1 & \sigma - p < i \le p \end{cases} \tag{A6.2}$$

where $\sigma$ is the weight of $M$ and $\sigma - p$ is the symbolic number of states of the given system.

Using these assumptions and the first assumption of the lemma, the system equations can be equivalently written as

$$\dot{y}_i = f_i(y, \dot{y}_{j \ne i}, u) \qquad 1 \le i \le \sigma - p \tag{A6.3a}$$

$$y_i = f_i(y_{j \ne i}, u) \qquad \sigma - p < i \le p \tag{A6.3b}$$

---

[1] The converse of this fact is true if we are allowed to increase the weights of edges in $B$ by using appropriate *ADDW* processes. The proof of this result is actually the key step in the proof of the lemma.

[2] This is because we can always renumber the equations and the variables to satisfy our assumptions.

where $\dot{y}_{j\neq i} \triangleq \{\dot{y}_j \mid j = 1,\ldots,i-1,i+1,\ldots,p\}$, $y_{j\neq i} \triangleq \{y_j \mid j = 1,\ldots,i-1,i+1,\ldots,p\}$ and $f_i$, $i = 1,2,\ldots,p$ are smooth functions. Note that the weighted bipartite graph of (A6.3) is *identical* to that of the original system equations. From here on, we use (A6.3) as the representation of the system and all differentiation steps that we specify are meant to be performed on (A6.3).

As demonstrated in the examples of Chapter 5, the basic step in the tranformation is to keep on differentiating appropriate equations of (A6.3) until the order of time derivative of each variable on the left hand side of (A6.3) is larger than or equal to any of its occurrences on the right hand side. This corresponds to applying appropriate *ADDW* procedures to the weighted bipartite graph $G = (S, V, B)$ of (A6.3) so as to make $M$ (as given by (A6.1)) a dominant matching of the resulting graph. After that, the differentiated system is converted into a first order canonical form by introducing additional state variables (see examples 5.2 and 5.3).

Before we give an algorithm for specifying appropriate *ADDW* procedures to make a maximum weighted complete matching a dominant complete matching, we need the following definitions.

$S_k \triangleq \{1,2,\ldots,k\}$, i.e., it contains the first $k$ nodes of $S$.

$V_k \triangleq \{1,2,\ldots,k\}$, i.e., it contains the first $k$ nodes of $V$.

$M_k \triangleq \{(1,1), (2,2), \ldots, (k,k)\}$. Hence $M_p = M$.

$G_k \triangleq (S_k, V_k, B_k)$ where $B_k \triangleq \{(i,j) \in B \mid i \in S_k, j \in V_k\}$.
$G_k$ is called the *subgraph* of $G$ *induced* by $(S_k, V_k)$.

$P(k,j)$ is a set of all *alternating paths* in $G_k$ with respect to $M_k$ from $k \in S_k$ to $j \in S_k$ such that the first edge in each path which is incident to $k$ must be a non-matching edge. Hence, $P(k,k)$ is a set of all alternating cycles that contain the edge $(k,k)$.

**Algorithm A.1** (convert a max. weighted complete matching to a dominant one)

For $k = \sigma-p+1, \sigma-p+2, \cdots, p$ {

compute $\psi = \max_i \{w(i,k) - w(k,k) \mid (i,k) \in B_k\}$ (A6.4)

if $\psi > 0$, $ADDW(\psi,k,G)$.

For $i = 1, 2, \cdots, k-1$ {

compute $\delta_{ki} = \max_{P \in P(k,i)} \{ \sum_{(l,m) \in P \cap M_k} w(l,m) - \sum_{(l,m) \in P \cap M_k} w(l,m) \}$. (A6.5)

}

For $i = 1, 2, \cdots, k-1$ {

if $\delta_{ki} > 0$, $ADDW(\delta_{ki},i,G)$

}

}

■

We shall prove by induction that, after applying Algorithm A.1, the maximum weighted complete matching $M$ becomes a dominant complete matching. First before we apply the algorithm, we have that

$$M_j \text{ is a dominant complete matching of } G_j \quad \text{for all } j \in [1,\sigma-p] \tag{A6.6}$$

Now, assume that the algorithm has been executed at $k = \hat{k} - 1$ and (A6.6) is true for all $j \in [1,\hat{k}-1]$. Suppose that, after the algorithm is executed at $k = \hat{k}$, (A6.6) is no longer true for all $j \in [1,\hat{k}]$. Then, there must exist $i,j \in [1,\hat{k}]$; $i \neq j$ such that

$$\tilde{w}(j,i) > \tilde{w}(i,i) \tag{A6.7}$$

where $\tilde{w}$ denotes the weight function after completing the algorithm at $k = \hat{k}$. From (A6.5) of the algorithm, we have

$$\tilde{w}(j,i) = \begin{cases} w(j,i) + \delta_{\hat{k}j} & \text{if } j < \hat{k} \\ w(j,i) & \text{if } j = \hat{k} \end{cases} \qquad \text{(A6.8)}$$

**Case 1** $(i = \hat{k})$

Applying (A6.7) and (A6.8), we have

$$w(j,\hat{k}) + \delta_{\hat{k}j} > w(\hat{k},\hat{k})$$

Substituting the value of $\delta_{\hat{k}j}$ in the above inequality, we obtain

$$w(j,\hat{k}) + \sum_{(l,m) \in \hat{P} \cap M_{\hat{k}}} w(l,m) - \sum_{(l,m) \in \hat{P} \cap M_{\hat{k}}} w(l,m) - w(\hat{k},\hat{k}) > 0 \qquad \text{(A6.9)}$$

where $\hat{P}$ is the maximizer of (A6.5) for $\delta_{\hat{k}j}$. Let

$$L = \hat{P} \cup (j,\hat{k}) \cup (\hat{k},\hat{k})$$

Then (A6.9) can be written as

$$\sum_{(l,m) \in L \cap M_{\hat{k}}} w(l,m) - \sum_{(l,m) \in L \cap M_{\hat{k}}} w(l,m) > 0 \qquad \text{(A6.10)}$$

But $L$ is an alternating cycle. Hence (A6.10) contradicts the maximum weight property of $M$ as given by Lemma 4.1.

**Case 2** $(i < \hat{k}$ and $j = \hat{k})$

Applying (A6.7) and (A6.8), we have

$$w(\hat{k},i) > w(i,i) + \delta_{\hat{k}i}$$

or

$$w(\hat{k},i) - w(i,i) > \delta_{\hat{k}i}$$

This clearly contradicts the maximality of $\delta_{\hat{k}i}$ since

$$\{(\hat{k},i), (i,i)\} \in P(\hat{k},i)$$

**Case 3** $(i < \hat{k}$ and $j < \hat{k})$

Applying (A6.7) and (A6.8), we have

$$w(j,i) - w(i,i) + \delta_{\hat{k}j} > \delta_{\hat{k}i} \qquad \text{(A6.11)}$$

Let $\hat{P} \in P(\hat{k},j)$ be a maximizer of (A6.5) for $\delta_{\hat{k}i}$. Since

$$\hat{P} \cup (j,i) \cup (i,i) \in P(\hat{k},i)$$

Therefore (A6.11) contradicts the maximality of $\delta_{\hat{k}i}$.

Since we obtain a contradiction in each of these three cases, we conclude that (A6.6) has to be true for all $j \in [1,\hat{k}]$, given that it is true for all $j \in [1,\hat{k}-1]$. Therefore, by induction, $M_p = M$ is indeed a dominant complete matching of $G$.

The remaining steps of the construction proof of the lemma are trivial and has been discussed earlier. Therefore the proof of this lemma is completed. ∎

The following facts are useful in the proof of Theorem 6.1

**Lemma A.2** Define $\|\cdot\|_\infty$ in $\mathbb{R}^n$ and $\mathbb{R}^{n\times n}$ as follows:

$$\|x\|_\infty \triangleq \max_{1\le i\le n} |x_i| \quad \text{where } x_i \text{ is the } i\text{-th component of } x \in \mathbb{R}^n, \text{ and}$$

$$\|A\|_\infty \triangleq \max_{1\le i\le n} \sum_{j=1}^{n} A_{ij} \quad \text{where } A_{ij} \text{ is the } (i,j)\text{-th element of } A \in \mathbb{R}^{n\times n}.$$

Let $L$ and $U$ be strictly upper and lower triangular matrices in $\mathbb{R}^{n\times n}$ such that $L\ge 0$, $U\ge 0$[1] and

$$\|L+U\|_\infty = \max_{1\le i\le n}\left\{\sum_{j=1}^{n}(L_{ij}+U_{ij})\right\} \le 1 \tag{A7.1}$$

where $L_{ij}$, $U_{ij}$ are the $(i,j)$-th elements of $L$, $U$ respectively. Then

$$\|(I-L)^{-1}U\|_\infty \le \|L+U\|_\infty \le 1$$

**Proof.** Let $\hat{x} = \text{col}(1,1,\ldots,1) \in \mathbb{R}^n$, then from (A7.1) we have

$$(I-L-U)\hat{x} \ge 0 \tag{A7.2}$$

Since $L$ is strictly lower triangular, we have

$$(I-L)^{-1} = I + L + L^2 + \cdots + L^{n-1}$$

Therefore

$$\begin{aligned}(L+U) - (I-L)^{-1}U &= [(I-L)^{-1} - I](I-L-U)\\ &= (L + L^2 + \cdots + L^{n-1})(I-L-U)\\ (L+U)\hat{x} - (I-L)^{-1}U\hat{x} &= (L + L^2 + \cdots + L^{n-1})[(I-L-U)\hat{x}]\end{aligned} \tag{A7.3}$$

(A7.2), (A7.3) and $L\ge 0$ imply that

$$(L+U)\hat{x} - (I-L)^{-1}U\hat{x} \ge 0 \tag{A7.4}$$

Since $(L+U) \ge 0$ and $(I-L)^{-1}U = (I + L + \cdots + L^{n-1})U \ge 0$, we have

---

[1] A vector $x$ or a matrix $A$ whose elements are all nonnegative is denoted by $x\ge 0$ or $A\ge 0$ respectively.

$$\|L+U\|_\infty = \|(L+U)\hat{x}\|_\infty \text{ and } \|(I-L)^{-1}U\|_\infty = \|(I-L)^{-1}U\hat{x}\|_\infty$$

Therefore (A7.4) implies that

$$\|L+U\|_\infty \geq \|(I-L)^{-1}U\|_\infty \qquad \blacksquare$$

**Lemma A.3** Let $f_1, f_2 : D \subseteq R^n \to R$ be two bounded[2] Lipschitz continuous functions, then the product function $f_1 f_2$ is also bounded Lipschitz continuous in D. If $f_2$ is also bounded away from zero, i.e., $\inf_{x \in D} |f_2(x)| = \bar{f}_2 > 0$, then $\frac{f_1}{f_2}$ is also bounded Lipschitz continuous in D.

**Proof.** Let $\hat{f}_1 = \sup_{x \in D} |f_1(x)|$, $\hat{f}_2 = \sup_{x \in D} |f_2(x)|$ and $\lambda_1, \lambda_2$ be the Lipschitz constants of $f_1, f_2$ respectively. Then for any $x, y \in D$

$$\begin{aligned}
|(f_1 f_2)(x) - (f_1 f_2)(y)| &= |f_1(x)f_2(x) - f_1(y)f_2(y)| \\
&= |[f_1(x) - f_1(y)]f_2(x) + [f_2(x) - f_2(y)]f_1(y)| \\
&\leq |f_1(x) - f_1(y)|\,|f_2(x)| + |f_2(x) - f_2(y)|\,|f_1(y)| \\
&\leq \lambda_1 \hat{f}_2 \|x-y\| + \lambda_2 \hat{f}_1 \|x-y\| \\
&= (\lambda_1 \hat{f}_2 + \lambda_2 \hat{f}_1)\|x-y\|
\end{aligned}$$

Therefore $f_1 f_2$ is Lipschitz continuous in D and is bounded by $\hat{f}_1 \hat{f}_2$.

Now, if $f_2$ is bounded away from zero, then

$$\left|\frac{1}{f_2(x)} - \frac{1}{f_2(y)}\right| = \left|\frac{f_2(y) - f_2(x)}{f_2(y)f_2(x)}\right| \leq \frac{\lambda_2}{\bar{f}_2^2}\|x-y\|$$

That is $\frac{1}{f_2}$ is Lipschitz continuous in D and is bounded by $\frac{1}{\bar{f}_2}$. Therefore by using the previous result, the product $f_1 \frac{1}{f_2} = \frac{f_1}{f_2}$ is also a bounded Lipschitz continuous function in D. $\blacksquare$

---

[2] If D is a bounded subset of $R^n$, then any function which is Lipschitz continuous in D is also bounded in D.

# Proof of Theorem 6.1

Define

$$L,\ U : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^r \to \mathbb{R}^{n \times n} \qquad f : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^r \to \mathbb{R}^n$$

$$H : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^r \to \mathbb{R}^{n \times n} \qquad \tilde{F} : \mathbb{R}^l \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^r \to \mathbb{R}^n$$

$$F : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^r \to \mathbb{R}^n \qquad \tilde{f} : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^l \times \mathbb{R}^r \to \mathbb{R}^n$$

as follows:

$$L_{ij}(a,b,u) \triangleq \begin{cases} 0 & ; \quad \text{if } i \le j \\ -\dfrac{C_{ij}(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u)}{C_{ii}(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u)} & ; \quad \text{if } i > j \end{cases}$$

$$U_{ij}(a,b,u) \triangleq \begin{cases} -\dfrac{C_{ij}(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u)}{C_{ii}(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u)} & ; \quad \text{if } i < j \\ 0 & ; \quad \text{if } i \ge j \end{cases}$$

$$H(a,b,u) \triangleq [I - L(a,b,u)]^{-1} U(a,b,u)$$

$$F_i(a,b,u) \triangleq -\frac{q_i(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u)}{C_{ii}(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u)}$$

$$f(a,b,s,u) \triangleq [I - L(a,b,u)]^{-1} F(a,b,u) + H(a,b,u)s$$

$$\tilde{F}_i(z,a,b,u) \triangleq -\frac{\tilde{q}_i(z,a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u)}{C_{ii}(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u)}$$

$$\tilde{f}(a,b,s,z,u) \triangleq [I - L(a,b,u)]^{-1} \tilde{F}(z,a,b,u) + H(a,b,u)s$$

where $C$, $q$, $\tilde{q}$ are previously defined in Algorithm 6.1 and Algorithm 6.2. Based on these definitions, Algorithm 6.1 can be tranformed into the following canonical form

$$\dot{v}^k = f(v^k, v^{k-1}, \dot{v}^{k-1}, u) \quad ; \quad v^k(0) = V$$

And Algorithm 6.2 can be tranformed into the following canonical form

$$\dot{v}^k = \tilde{f}(v^k, v^{k-1}, \dot{v}^{k-1}, z^{k-1}, u) \; ; \quad v^k(0) = V$$

$$z^k = g(v^k, u)$$

From Lemma A.3 and the assumptions of Theorem 6.1, we can deduce that

a) $f$, $\tilde{f}$, $g$ are continuous functions and are Lipschitz continuous with respect to $v^k$ and $v^{k-1}$.

b) $\tilde{f}$ is also Lipschitz continuous with respect to $z$, i.e. there is a constant $\lambda>0$ such that

$$\|\tilde{f}(a, b, s, z, u) - \tilde{f}(a, b, s, \tilde{z}, u)\|_\infty \leq \lambda\|z - \tilde{z}\|_\infty \tag{A8.1}$$

where $\|\cdot\|_\infty$ denotes the standard max-norm as defined in Lemma A.2.

Applying the result of Lemma A.2, we have

$$\begin{aligned}\|H(a,b,u)\|_\infty &\leq \|L(a,b,u) + U(a,b,u)\|_\infty \\ &= \max_{1\leq i\leq n} \frac{-\sum_{\substack{j=1\\ j\neq i}}^{n} C_{ij}(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u)}{C_{ii}(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u)}\end{aligned} \tag{A8.2}$$

From the definition of $C$ and the assumptions of Theorem 6.1, we have

$$C_{ii}(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u) \leq C_{\min} - \sum_{\substack{j=1\\ j\neq i}}^{n} C_{ij}(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u) \tag{A8.3}$$

and

$$-\sum_{\substack{j=1\\ j\neq i}}^{n} C_{ij}(a_1, \ldots, a_i, b_{i+1}, \ldots, b_n, u) \leq (n-1)C_{\max} \tag{A8.4}$$

Since, for all $\sigma \leq 0$, the function $\frac{\sigma}{C_{\min} + \sigma}$ is monotonically increasing with respect to $\sigma$, therefore (A8.2), (A8.3) and (A8.4) imply that

$$\|H(a,b,u)\|_\infty \leq \frac{(n-1)C_{\max}}{C_{\min} + (n-1)C_{\max}} = \gamma < 1 \tag{A8.5}$$

Therefore Algorithm 6.1 satisfies all the conditions of Theorem 5.3 and hence it converges for any piecewise continuous input $u$.

For Algorithm 6.2, we define $\|\cdot\|$ in $\mathbf{R}^n \times \mathbf{R}^l$ such that for any $s \in \mathbf{R}^n$ and $z \in \mathbf{R}^l$

$$\left\| \begin{matrix} s \\ z \end{matrix} \right\| \triangleq \|s\|_\infty + \frac{\lambda}{\gamma}\|z\|_\infty$$

where $\lambda$ is as given in (A8.1) and $\gamma$ is as given in (A8.5). Then

$$\begin{aligned} \left\| \begin{matrix} \tilde{f}(a, b, s, z, u) - \tilde{f}(a, b, \tilde{s}, \tilde{z}, u) \\ g(a, u) - g(a, u) \end{matrix} \right\| &= \|\tilde{f}(a, b, s, z, u) - \tilde{f}(a, b, \tilde{s}, \tilde{z}, u)\|_\infty \\ &\leq \lambda\|z - \tilde{z}\|_\infty + \|H(a,b,u)\|_\infty\|s - \tilde{s}\|_\infty \\ &\leq \gamma\left[\|s - \tilde{s}\|_\infty + \frac{\lambda}{\gamma}\|z - \tilde{z}\|_\infty\right] \\ &= \gamma\left\| \begin{matrix} s - \tilde{s} \\ z - \tilde{z} \end{matrix} \right\| \end{aligned}$$

Therefore Algorithm 6.2 also satisfies the conditions of Theorem 5.3 and hence it converges for any piecewise continuous input $u$. ∎

# Appendix B

## Applications of Iterative Nonlinear Relaxation Methods in Time Domain Simulation of MOS Circuits

It has been shown [18] that the use of non-iterative (i.e., only one iteration) nonlinear relaxation methods in timing simulation of MOS circuits, as implemented in MOTIS [6], MOTIS-C [27] and DIANA [9], has deteriorated the numerical properties (i.e., stability and accuracy) of the integration method used in the time discretization of the circuit differential equations. Since there are no additional computational steps to verify that the nonlinear equations are solved accurately enough at every timepoint, the estimation of the local truncation error of the integration method is no longer reliable. Hence the timestep control mechanism based on local truncation errors is no longer valid for timing simulation.

We now discuss the use of conventional iterative nonlinear relaxation methods in time domain solution. The basic idea is to solve the nonlinear equations at each timepoint iteratively by a relaxation method until satisfactory convergence is achieved. Hence, the numerical properties of the integration method are retained and the estimation of the local truncation error can be used as a factor in determining the sizes of the timesteps commensurated with the accuracy requirement. The main problem to be addressed here is whether or not the relaxation iteration will always converge. We will show that, for MOS circuits with the conventional assumption that there is a grounded capacitor to every node, there exists a minimum timestep $h_{min}$ which guarantees the convergence of the relaxation iteration. The size of $h_{min}$ is of course dependent on how the circuit is decomposed. We then give a simplified timestep control mechanism which combines the uses of the local truncation error and the relaxation iteration count in

determining the sizes of the timesteps.

Consider an MOS circuit whose node equations can be written as

$$C\dot{v} - f(v, u) = 0 ; \qquad v(0) = V \tag{B.1}$$

where $v(t) \in \mathbf{R}^n$ is the vector of unknown node voltages, $u(t) \in \mathbf{R}^r$ is the vector of independent sources, $f : \mathbf{R}^n \times \mathbf{R}^r \rightarrow \mathbf{R}^n$ is a Lipschitz continuous function each component of which represents the sum of currents feeding the capacitors at each node and $C \in \mathbf{R}^{n \times n}$ is a symmetric strictly diagonally dominant matrix. Applying the Backward Euler integration method to (B.1), we obtain the following discrete time sequence of nonlinear equations

$$C(v_{i+1} - v_i) - h_{i+1} f(v_{i+1}, u_{i+1}) = 0 ; \qquad v_0 = V \tag{B.2}$$

where $h_{i+1} = t_{i+1} - t_i$, $u_{i+1} = u(t_{i+1})$, $v_i \approx v(t_i)$ and $v_{i+1} \approx v(t_{i+1})$. For the sake of simplicity, we consider the pointwise relaxation method only. Let

$$C = D + L + U \tag{B.3}$$

where $D \in \mathbf{R}^{n \times n}$ is a diagonal matrix, $L \in \mathbf{R}^{n \times n}$ is a strictly lower triangular matrix and $U \in \mathbf{R}^{n \times n}$ is a strictly upper triangular matrix.

Applying the iterative nonlinear pointwise Gauss-Jacobi [17] relaxation method to (B.2), we obtain the following GJ iterative equation

$$D(v_{i+1}^{GJ^k}) + (L + U)v_{i+1}^{GJ^{k-1}} - C v_i - h_{i+1} f_{GJ}(v_{i+1}^{GJ^k}, v_{i+1}^{GJ^{k-1}}, u_{i+1}) = 0 \tag{B.4}$$

where $v_{i+1}^{GJ^{k-1}} \in \mathbf{R}^n$, $v_{i+1}^{GJ^k} \in \mathbf{R}^n$, $k$ is the iteration count and, for each component index $j = 1,2,\ldots,n$,

$$f_{GJ_j}(v_{i+1}^{GJ^k}, v_{i+1}^{GJ^{k-1}}, u_{i+1}) \triangleq f_j(v_{i+1_1}^{GJ^{k-1}}, \ldots, v_{i+1_{j-1}}^{GJ^{k-1}}, v_{i+1_j}^{GJ^k}, v_{i+1_{j+1}}^{GJ^{k-1}}, \ldots, v_{i+1_n}^{GJ^{k-1}}, u_{i+1}) \tag{B.5}$$

The initial guess for starting (B.4) is the Forward Euler predictor, i.e.,

$$v_{i+1}^{GJ^0} = v_i + \frac{h_{i+1}}{h_i}(v_i - v_{i-1}) \tag{B.6}$$

Applying the iterative nonlinear pointwise Gauss-Seidel (GS) [17] relaxation method to (B.2), we obtain the following GS iterative equation

$$(D + L)v_{i+1}^{GS^k} + Uv_{i+1}^{GS^{k-1}} - Cv_i - h_{i+1}f_{GS}(v_{i+1}^{GS^k}, v_{i+1}^{GS^{k-1}}, u_{i+1}) = 0 \tag{B.7}$$

where $v_{i+1}^{GS^{k-1}} \in \mathbf{R}^n$, $v_{i+1}^{GS^k} \in \mathbf{R}^n$, $k$ is the iteration count and, for each component index $j = 1,2,\ldots,n$,

$$f_{GS_j}(v_{i+1}^{GS^k}, v_{i+1}^{GS^{k-1}}, u_{i+1}) \triangleq f_j(v_{i+1_1}^{GS^k}, \ldots, v_{i+1_j}^{GS^k}, v_{i+1_{j+1}}^{GS^{k-1}}, \ldots, v_{i+1_n}^{GS^{k-1}}, u_{i+1}) \tag{B.8}$$

The initial guess for starting (B.7) is the Forward Euler predictor, i.e.,

$$v_{i+1}^{GS^0} = v_i + \frac{h_{i+1}}{h_i}(v_i - v_{i-1}) \tag{B.9}$$

We now state the following result which states that these two iterative relaxation methods will always converge independent of the initial guess provided that the stepsize $h_{i+1}$ is sufficiently small.

**Theorem B.1** There exists $h_{\min}$ such that if $h_{i+1} \leq h_{\min}$ then the sequence of iterated solutions of either the pointwise GJ iterative equation (B.4) or pointwise GS iterative equation (B.7) converges to the solution of (B.2)

**Proof** From (B.4), we have

$$(v_{i+1}^{GJ^k} - v_{i+1}^{GJ^{k-1}}) = -D^{-1}(L + U)(v_{i+1}^{GJ^{k-1}} - v_{i+1}^{GJ^{k-2}}) + \tag{B.10}$$
$$h_{i+1}D^{-1}[f_{GJ}(v_{i+1}^{GJ^k}, v_{i+1}^{GJ^{k-1}}, u_{i+1}) - f_{GJ}(v_{i+1}^{GJ^{k-1}}, v_{i+1}^{GJ^{k-2}}, u_{i+1})]$$

Let

$$E_{GJ}^k \triangleq \|v_{i+1}^{GJ^k} - v_{i+1}^{GJ^{k-1}}\|_{GJ} \tag{B.11}$$

and $L_{GJ}$ be the Lipschitz constant of $f$ with respect to $\|\cdot\|_{GJ}$. Then from (B.10), we have

$$E^k_{GJ} \leq \|D^{-1}(L+U)\|_{GJ} E^k_{GJ^{-1}} + h_{i+1}\|D^{-1}\|_{GJ} L_{GJ}(E^k_{GJ} + E^k_{GJ^{-1}})$$

which yields ( assuming that $h_{i+1}\|D^{-1}\|_{GJ} L_{GJ} < 1$ )

$$E^k_{GJ} \leq \frac{\gamma_{GJ} + h_{i+1}\|D^{-1}\|_{GJ} L_{GJ}}{1 - h_{i+1}\|D^{-1}\|_{GJ} L_{GJ}} E^k_{GJ^{-1}}$$

$$\triangleq \tilde{\gamma}_{GJ}(h_{i+1}) E^k_{GJ^{-1}} \tag{B.12}$$

where

$$\gamma_{GJ} \triangleq \|D^{-1}(L+U)\|_{GJ} \tag{B.13}$$

Therefore, the GJ iterated solutions of (B.4) will converge if $\tilde{\gamma}_{GJ}(h_{i+1}) < 1$.

Similarly, for the pointwise GS iterative equation (B.7), we have

$$E^k_{GS} \leq \frac{\gamma_{GS} + h_{i+1}\|D^{-1}\|_{GS} L_{GS}}{1 - h_{i+1}\|D^{-1}\|_{GS} L_{GS}} E^k_{GS^{-1}}$$

$$\triangleq \tilde{\gamma}_{GS}(h_{i+1}) E^k_{GS^{-1}} \tag{B.14}$$

where $L_{GS}$ is the Lipschitz constant of $f$ with respect to $\|\cdot\|_{GS}$,

$$\gamma_{GS} \triangleq \|(D+L)^{-1}U\|_{GS} \tag{B.15}$$

and

$$E^k_{GS} \triangleq \|v^{GS^k}_{i+1} - v^{GS^{k-1}}_{i+1}\|_{GS} \tag{B.16}$$

Hence, the GS iterated solutions of (B.7) will converge if $\tilde{\gamma}_{GS}(h_{i+1}) < 1$.

Since $C$ as defined in (B.1) is symmetric and strictly diagonally dominance, it can be shown ( see [17] ) that there exist $\|\cdot\|_{GJ}$ and $\|\cdot\|_{GS}$ such that

$$\gamma_{GJ} < 1 \qquad \text{and} \qquad \gamma_{GS} < 1$$

From (B.12), choose $h_{i+1} = h_{GJ}$ such that

$$\tilde{\gamma}_{GJ}(h_{GJ}) = \frac{\gamma_{GJ} + h_{GJ}\|D^{-1}\|_{GJ} L_{GJ}}{1 - h_{GJ}\|D^{-1}\|_{GJ} L_{GJ}} = \frac{1+\gamma_{GJ}}{2} < 1 \tag{B.17}$$

which yields

$$h_{GJ} = \frac{1 - \gamma_{GJ}}{(3 + \gamma_{GJ})\|D^{-1}\|_{GJ} L_{GJ}} \tag{B.18}$$

It is then easy to show that

$$\tilde{\gamma}_{GJ}(h_{i+1}) \leq \tilde{\gamma}_{GJ}(h_{GJ}) \qquad \text{for all} \quad h_{i+1} \leq h_{GJ} \tag{B.19}$$

Similarly, for (B.14), letting

$$h_{GS} = \frac{1 - \gamma_{GS}}{(3 + \gamma_{GS})\|(D + L)^{-1}\|_{GS} L_{GS}} \tag{B.20}$$

we have that

$$\begin{aligned} \tilde{\gamma}_{GS}(h_{i+1}) &\leq \tilde{\gamma}_{GS}(h_{GS}) = \frac{1 + \gamma_{GS}}{2} \\ &< 1 \qquad \text{for all} \quad h_{i+1} \leq h_{GS} \end{aligned} \tag{B.21}$$

The proof is then completed by choosing

$$h_{\min} = \min(h_{GJ}, h_{GS}) \qquad \blacksquare$$

We now propose a simple timestep control mechanism for time domain simulation using an iterative nonlinear relaxation method as shown below. The basic idea is to cut the size of the timestep automatically when the relaxation iteration does not converge within a prescribed number of iterations.

**Algorithm B.1 (Timestep Control Mechanism for Nonlinear Relaxation Method)**

Data $h_1$ = initial timestep, $N$ = maximum number of relaxation iterations after which monotonic decreasing of the iteration error is expected.

Step 0: Set $i = 0$, $t_1 = h_1$.

Step 1: Set $k = 1$ and $v_{i+1}^0$ = the predicted solution at time $t_{i+1}$.

Step 2: Apply one iteration of relaxation method to solve the nonlinear equations at time $t_{i+1}$ for $v_{i+1}^k$.

Step 3: If $\|v_{i+1}^k - v_{i+1}^{k-1}\| \leq \delta$ where $\delta$ is a predetermined relaxation convergence error {

Set $v_{i+1} = v_{i+1}^k$ and compute $h_{i+2}$ based on the estimation of the local truncation error (LTE).

If ( LTE is too large ) then set $h_{i+1} = h_{i+2}$.

Else set $i = i + 1$.

Go to Step 1.

}

Else if $(k < N)$ then set $k = k+1$ and go to Step 2.

Else if $\|v_{i+1}^k - v_{i+1}^{k-1}\| < \|v_{i+1}^{k-1} - v_{i+1}^{k-2}\|$ then set $k = k+1$ and go to Step 2.

Else set $h_{i+1} = \frac{h_{i+1}}{2}$ and go to Step 1. ∎