ELOGIC:

A RELAXATION-BASED SWITCH-LEVEL SIMULATION TECHNIQUE

by

Young Hwan Kim

Memorandum No. M86/2

3 December 1985

Cover page

ELOGIC:

A RELAXATION-BASED SWITCH-LEVEL SIMULATION TECHNIQUE

by

Young Hwan Kim

Memorandum No. M86/2

3 December 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

Electrical-Logic (ELogic) is a new form of relaxation-based switch-level modeling and simulation which provides more accurate timing information than existing strength-oriented logic simulation or switch-level simulation, at almost the same speed. ELogic can be used for circuit simulation, mixed-mode simulation and as the basis for accurate, switch-level timing verification of MOS digital circuits.

In this report, the basic ELogic algorithm and two of its derivatives are presented and described in detail so that the appropriate algorithm can be chosen for a particular application. The description of the implementation and results of the experiments are included. The application of ELogic algorithms to both simulation and timing verification are also included.

Acknowledgements

I sincerely would like to thank my research advisor Prof. A. Richard Newton for his consistent encouragement and effective guidance throughout course of my work.

I also wish to thank everyone in the CAD group at University of California,Berkeley, who has provided me with the privilege to work in such a good co-operative atmosphere. But a few people deserve special mention. I would like to thank Resve A. Saleh for the many long hours of engagement in useful discussion of the algorithms and their implementation. I also thank him for his critical reading of the manuscript. The implementation of ELogic technique as a delay model in Crystal and all the related experiments are due mainly to the efforts of Seung Ho Hwang. Many bugs in the stand-alone program were corrected through discussion with him. I would also like to thank J. Kleckner for his initial work on ELogic and for the suggestion to work on it.

Finally, I wish to thank my wife, Young, for her patience and my family members in Korea for their support.

The support of SRC (Semiconductor Research Cooperation) is also acknowledged.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

Computer-aided integrated circuit simulation has been used extensively over the last few years for designing circuits, analyzing their electrical performance and verifying their function. Previously, a designer could synthesize a breadboarded version of the network with minimal effort, and make modifications to get a final version on the test bench. However MOS integrated circuits designed today contain thousands of transistors and form the basis for most large-scale integrated digital circuits. Therefore it is almost impossible to test the performance of these circuits by breadboarding. Another option is to fabricate prototype versions of the circuit and test them. However the fabrication of the circuit is very expensive and time consuming, and is not a feasible alternative for the debugging process. Therefore integrated circuit designers must use computer-aided design and analysis tools to ensure that their circuits will function correctly and meet the performance specifications rather than fabrication or breadboarding.

Table 1.1 illustrates the various levels of the integrated circuit simulation, their use and examples of the simulators for each level [1, 2]. Depending on the stage of the design process, the designer can select a simulator at the proper level.

Level	Use	Simulator Examples
Behavioral	Algorithmic Verification	GASP,SIMULA,ISPS,ADLIB
RTL	Logic Verification	ISPS,ADLIB,SPLICE2
Logic	Logic Verification	LOGIS,ILOGS,SPLICE2
Circuit	Performance Evaluation	SPICE2,ASTAP,SPLICE2
Device	Device Model Development	GEMINI
Process	Process Development	SUPREM,SAMPLE

Table 1.1 Hierarchy of Large Integrated Circuit Simulation

1.1. Motivation of Electrical-Logic Simulation

Circuit simulators, such as SPICE2 [3] and ASTAP [4], have been successfully used for the design and performance evaluation of integrated circuits, since they provide very accurate output waveforms. However, it is almost prohibitive to use these simulators for the analysis of large integrated circuits, due to long computer run times. A large effort has been made to reduce the required CPU-time for large circuits, while maintaining the same waveform accuracy. Techniques include the use of relaxation methods such as Iterated Timing Analysis(ITA) [2, 13, 5] and Waveform Relaxation(WR) [6, 32, 33] , and the use of vector processors [7, 8]. Relaxation methods provide significant speed improvement with the same waveform accuracy as SPICE2, assuming identical devices models, and have guaranteed convergence and stability properties [5]. The cost estimates of logic, relaxation-based and circuit simulation for simulating one 32-bit integer multiply instruction of a recent 450,000 device microprocessor [9] are shown in Table 1.2. The cost estimates are normalized to the performance of a logic level simulation on the IBM 370/168 and ignore virtual machine overhead i.e., assumes uniform addressing and memory access times [2, 10]. The term circuit simulation refers to the use of direct methods as implemented in the program SPICE2 [3, 11].

Table 1.2 indicates that relaxation-based simulation using ITA can be much faster (360 times) than a classical circuit simulation like SPICE2, with comparable waveform accuracy, for circuits that are loosely coupled. However ITA is still much slower (72 times) than logic simulation and the actual CPU-time requirement of relaxation-based simulation depends on the characteristic of the circuit under analysis. That is, strong coupling between nodes due to

Simulation	Normalized CPU-time
Logic	1
Relaxation-based(ITA)	72
Circuit	26000

Table 1.2 Normalized CPU-time Comparison

gain and feedback in the circuit may increase the simulation time unless some partitioning scheme [12] is used. As an extreme example, ITA without using any partitioning scheme may be slower than SPICE2 for analog circuits such as operational amplifiers, because they usually contain large feedback paths and high gain and there is little or no latency to be exploited [13].

One of the major differences between accurate electrical simulation(including ITA and WR) and logic simulation is the models used to describe the elements being analyzed. Logic simulators use much simpler models than circuit simulators. In circuit simulation, individual transistors are modeled using complex analytical expressions. However in a logic simulation, the circuits are modeled as logic gates, which are groups of transistors, or by single transistors modeled as ideal switches [14]. Therefore a logic simulator is used for functional verification or to obtain first-order timing information of integrated circuits, and therefore trades off accuracy in favor of shorter run times.

What is required is a simulation technique which fills the gap between logic simulation and circuit simulation, in terms of speed and accuracy. Electrical-logic (ELogic) simulation [15] is a new *relaxation-based switch-level* simulation technique which provides a continuous speed-accuracy tradeoff between the two simulation levels using transistor level circuit models. ELogic also can be used for any purpose where a continuous speed-accuracy tradeoff is desirable, for example, delay modeling in timing analysis. This is the motivation of electrical-logic simulation.

In Chapter 2, the basic ELogic algorithm and its two derivatives are described in detail. The best algorithm among the three depends on the application. Therefore, rather than proposing the best one, the detailed basis of the all algorithms are presented so that the one which best fits the purpose can be chosen and they can be modified if necessary. The description of how ELogic computes the output waveform and the comparison with existing relaxation-based electrical simulators are also included. In Chapter 3 and 4, the application of ELogic in simulation and as a delay modeling technique in timing verification are described. Conclusions are given in Chapter 5.

CHAPTER 2

ELECTRICAL-LOGIC SIMULATION

2.1. Introduction

Electrical-Logic (ELogic) simulation [2,15] is a new form of relaxation-based, multi-signal and multi-strength switch-level simulation which obtains waveforms in the circuit being analyzed. The notion of ***signal - strength*** of ELogic is similar to that of most modern logic simulators which allows transistors in the circuit description, in addition to logic gates [5,16,17,14,18]. This notion is included to handle the problems specific to MOS integrated circuits. In logic simulators, the most frequently used logic levels are 0 (FALSE), X (UNKNOWN) and 1 (TRUE). In ELogic, we define a set of discrete voltage levels called ***voltage states***. They can be considered as same as logic levels of logic simulations, except that an actual voltage value is assigned to each state. The strength used in logic simulation is an abstraction of conductance, not a specific value, from a node to ground or to power supply. It can be an attribute of MOS gate or a node. It is used to determine which element will decide the logic state ***(signal - strength pair)*** of the node, when many fanin logic-elements are connected at the same node. On the other hand, the strength of ELogic is defined for each circuit element with a specific conductance value and is used to obtain the waveforms of the circuits.

In ELogic, circuit nodes are allowed to make a transition from one state to an adjacent state only. Figure 2.1 represents an expected, ideal ELogic output waveform. In the Figure, the voltages of V0 through V4 are defined as a set of discrete voltage states. The dashed line represents an exact output waveform, while the solid line represents an ideal ELogic output waveform.

ELogic solves for the amount of time required to make a transition from one voltage state to an adjacent state, rather than solving for the amount of voltage change at each node

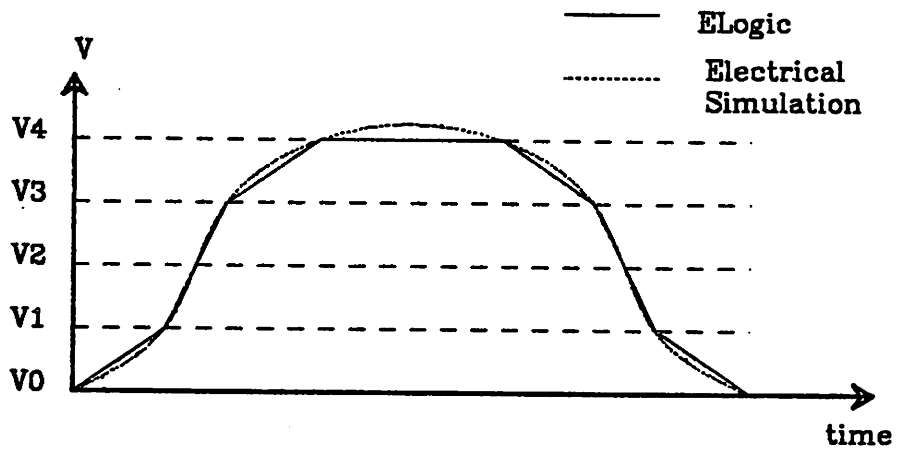


Fig. 2.1 Ideal ELogic Output Waveform

to the given time step as in conventional electrical simulators.

Figure 2.2 is an example of ELogic output waveforms at two neighboring nodes. Notice that the nodes achieve the next voltage state at different time points, independent of each other. The input to a circuit consists of a set of state changes at discrete time points. That *waveform* fragment is used as input to the state machine model of the input device and an output *waveform* fragment is computed. As a result, small edge transitions are moved through the network. Therefore ELogic can be considered as an asynchronous windowed waveform relaxation method, but can be implemented using the event-driven mechanism available in a relaxation-based simulator like SPLICE [2,31].

In this chapter, a basic ELogic technique will be described first, then the description of its versions will follow.

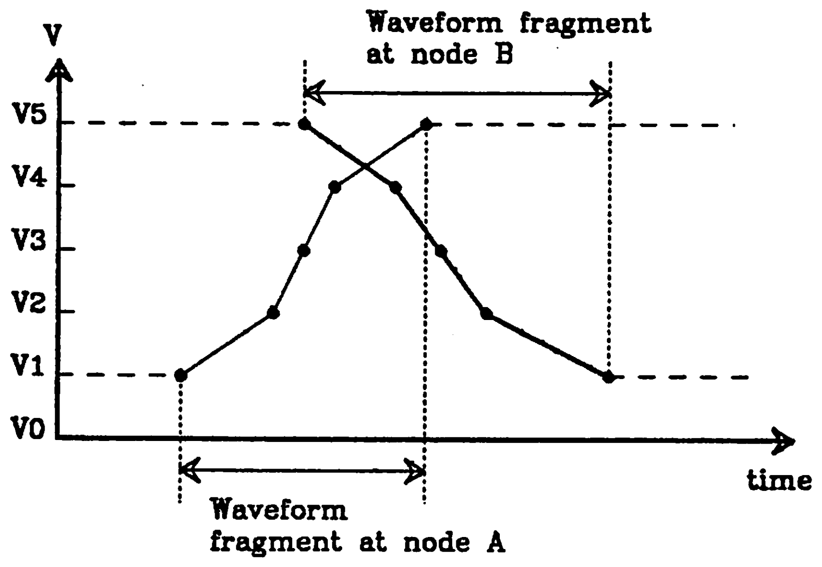
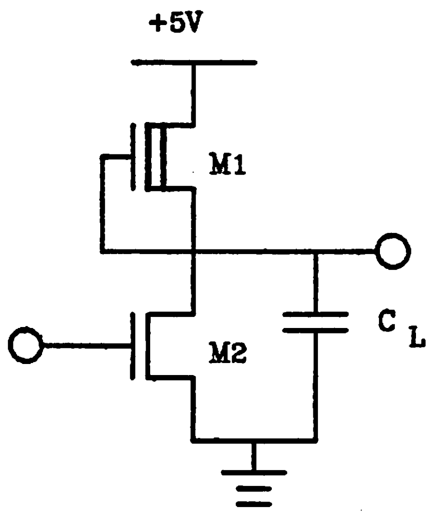


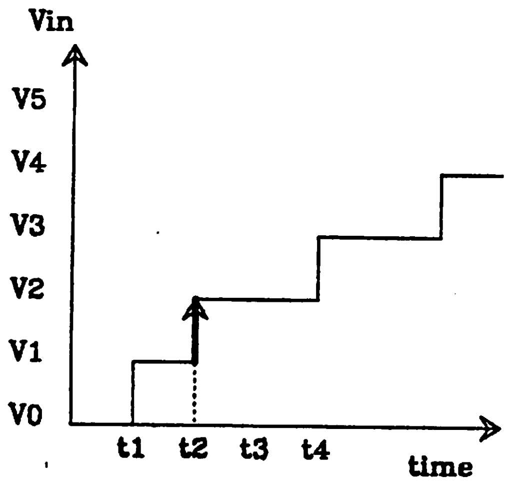
Fig. 2.2 Example of Output Waveforms at Two Neighboring Nodes illustrating asynchronous windowed waveform nature of ELogic

2.2. Inverter Example

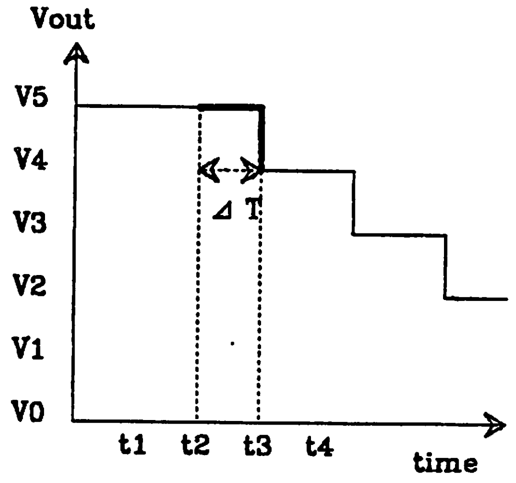
To illustrate further how ELogic works, consider an NMOS inverter of Figure 2.3(a). ELogic assumes that there is a grounded capacitor at each node as in other relaxation-based electrical simulators. Figures 2.3(b) and 2.3(c) represent an input waveform to the inverter and output waveform, respectively, where *V0, V1, V2, V3, V4* and *V5* are chosen as the discrete states using a uniform voltage step size. The input makes a sequence of transitions from *V0* to *V5* and the output makes a corresponding sequence of transitions from *V5* to *V0*. For example, the input waveform achieves a new voltage state *V1* from *V0*, at time t_1. However the input state *V1* is not high enough to pull down the output of the inverter and therefore it remains high. When the input reaches its new voltage state *V2* at time t_2, from *V1*, the output begins to pull down from *V5* to its new state *V4* and ELogic calculates the time



(a) NMOS Inverter



(b) Input Waveform



(c) Output Waveform

Fig. 2.3 NMOS Inverter Example using ELogic

required for that transition (ΔT of Figure 2.3(c)), using the electrical properties of the NMOS inverter. Sometimes the output node may be pulled one more voltage step down to *V3* and, if that is the case, ELogic calculates the time for that transition too. In this example, the output stays at *V4*, and when the input reaches the next new state *V2* at time t_4, the process continues. To summarize, ELogic determines whether or not the output can change its state in following two cases :

1) when the input reaches the new state
2) when the output reaches the new state

and if it changes, ELogic calculates the corresponding transition time.

2.3. Voltage States

The voltage states which ELogic nodes can take are a set of discrete values. If an ELogic node voltage is computed to lie between two adjacent states at some steady-state value, it must be rounded off to the nearest discrete ELogic state. Therefore the precision of an ELogic waveform is determined by the number and values of the voltage states defined.

The number and/or the values of the voltage states can be varied. Most of CPU-time saving of logic simulation over electrical simulation comes from fewer time points evaluated rather than fewer iterations at the same time point. From the previous section, it is observed that the number of time points evaluated during a transition depends on the number of voltage states. If more states are allowed for a simulation, the output waveform will be more precise, but it will cost more. On the other hand, if the number of the voltage states is reduced, it will cost less. However the output waveform will be less precise. Therefore ELogic can save CPU-time by trading off the waveform precision. Furthermore, since there is no restriction in defining the number and/or the values of the voltage states, ELogic can provide a continuous speed-accuracy trade-off. This is one of the major features of ELogic technique, in its applications. Sometimes a designer may need to quickly verify the logic function or determine the first order timing information of a preliminary circuit design. This usually happens at the early phase of a design process. In this case, only a few states would be defined for fast simulation speed. Later on, more states may be defined if more accurate information is desired. Using ELogic, a designer doesn't have to switch back and forth between a circuit simulator and a logic simulator. Also, different parts of the circuit may have a different set of ELogic states which constitutes a form of mixed-mode simulation.

The values of the voltage states may be nonuniform. Even though the same number of voltage states are allowed for the simulation of the same circuit, if different values of the states are chosen, the accuracy of the output waveform may be different. Therefore the user can choose an intelligent set of voltage states, based on the design and simulation experience.

2.4. Output Waveform Representation

The output waveforms generated from an ELogic simulation must represent all the information available without misleading the designer. Assume that a node voltage is *V1* at time *t1* and achieves a new voltage state *V2* at time *t2*. Some of the possible waveform representations are shown in Figure 2.4. Figure 2.4(a) represents the waveform by a straight line between *V1* and *V2*. In practice, this method provides detailed information about the starting and ending points of the transition. The disadvantage is that it may mislead the user, since it looks similar to the output waveform of an electrical simulator. A user may mistake it for an accurate output waveform which is obtained by electrical simulation. This is especially true when many voltage states are allowed. Figure 2.4(b) represents the waveform by a straight line at *t2*, with an infinite slope. Figure 2.4(c) uses a similar straight line, but in the middle of *t1* and *t2*. Although either method is less likely to mislead the user, they do not provide information about the start and/or end-time point of the transition. Another method is to use a string of numbers and characters to represent the output waveform rather than using a plot, as in Figure 2.4(d). In this figure, *V1* and *V2* are level numbers rather than specific voltage values and *R* means it is rising. This method gives all available timing information and does not mislead the user. However it is difficult to see the

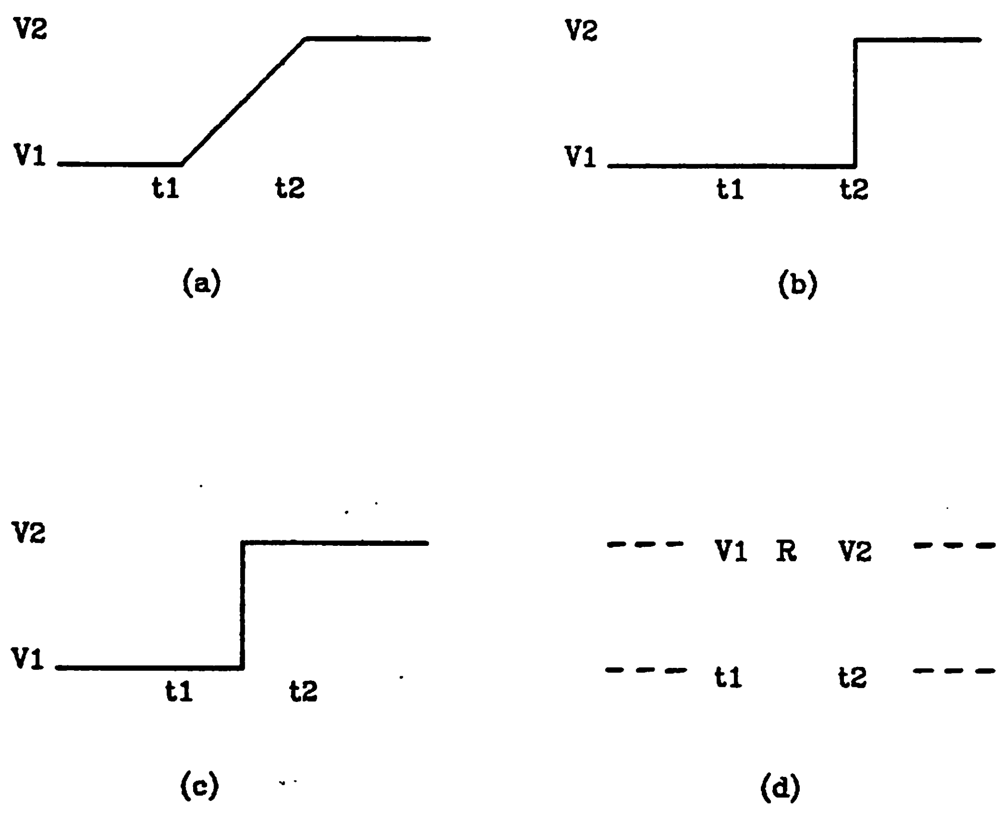


Fig. 2.4 Representation of Output Waveform

overall waveforms, particularly when the circuit being analyzed has waveforms which are analog in nature. Unless otherwise specified, Figure 2.4(a) will be used throughout this report to represent ELogic output waveforms.

2.5. Application of ELogic

The ELogic technique can be used in a variety of applications where a continuous accuracy-speed trade-off is useful.

First, ELogic can be used for simulation purposes. The waveform accuracy depends on the number and values of the voltage states used. Second, ELogic can be used as a basis for accurate critical path analysis for timing verification [19,20]. In general, the delay modeling section of a timing verifier is separate from a path analysis section in the program. By replacing an existing delay modeler by ELogic delay modeling, the timing verifier can provide more accurate critical path information, depending on the number of states used. Third, ELogic can be used for mixed-mode simulation. A mixed-mode simulator is one which allows more than one simulation level of Table 1.1 in the same simulation. In this report, the term mixed-mode simulation will be used to mean a simulation technique which allows an electrical simulation on part of the circuit and a logic simulation for the rest of the circuit depending on the required accuracy [2,21,22]. ELogic can be used to divide the circuit into as many sub-circuits as necessary, each with a different state model, depending on the required accuracy in that sub-circuit and also for the interface between them. That is, rather than dividing it into only two sub-circuits for an electrical simulation and a logic simulation, a mixed-mode simulation can also have a variety of ELogic state models. Furthermore, ELogic technique can be used for a basis of *electrical - logic* interface element in mixed-mode simulators. Since electrical simulation handles voltage and current while logic simulation handles logic levels, mixed-mode simulation generally requires a special interface element between two levels. A Boolean-controlled switch which is used in the program DIANA [21] is one example of Logic-to-Electrical signal conversion element. While pre-existing interface elements are used successfully, ELogic can provide a more accurate basis for the mixed-mode interface element. Finally, ELogic can be used as a basis for electrical fault simulation. Although a large effort has gone into developing logic fault simulators, electrical fault simulation is still an open research area. Certainly, one of the most important requirements of a simulation technique for electrical fault simulation is speed with moderate accuracy of waveforms and this is what ELogic can provide by choosing a proper set of voltage states.

In Chapter 3, the application of ELogic to simulation is described, and in Chapter 4, application to timing verification is presented.

2.6. ELogic Circuit Models

2.6.1. ELogic MOS Model

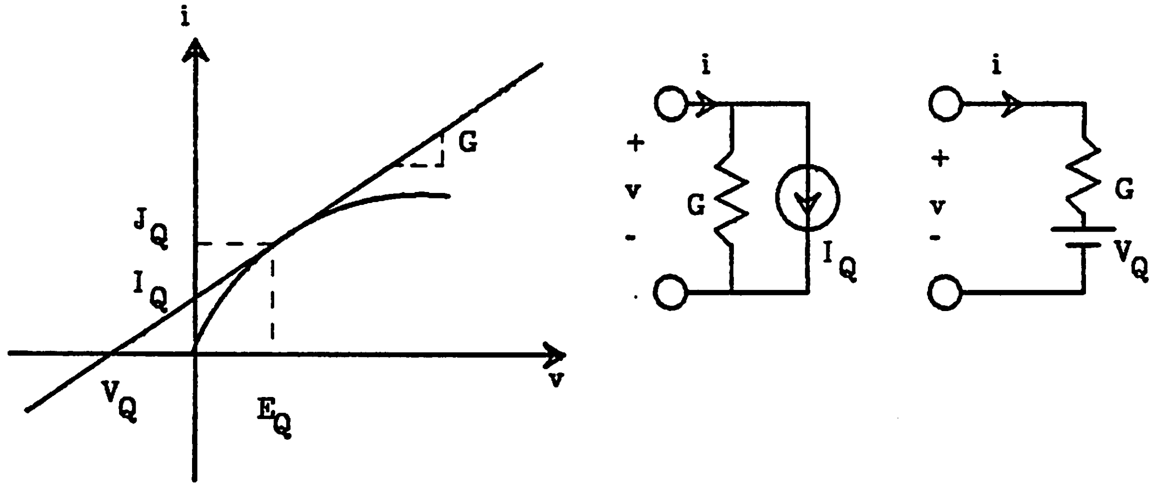
ELogic models MOS transistors using the Schichman-Hodges model [23] equations which are adequate for most MOS digital circuit simulation, provided that the dc parameters are obtained experimentally. Unless otherwise specified, the reference direction of terminal current of MOS transistors is defined as follows throughout this report :

Definition 1 : The sign of current is positive if it enters the MOS transistors.

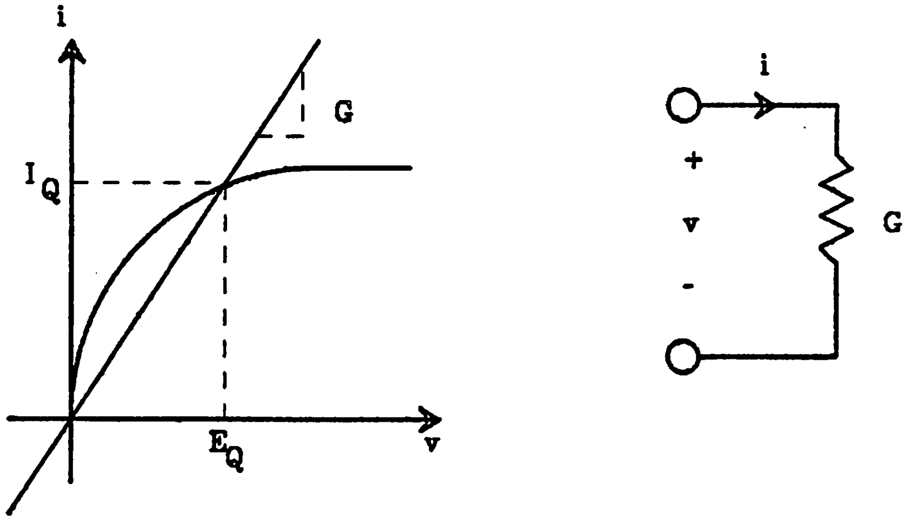
The Schichman-Hodges model equations for NMOS and PMOS transistors are shown in Table 2.1, where,

V_{TO} : zero-bias threshold voltage.
$KP = \mu C_{ox}$, process transconductance parameter.
ϕ_F : equilibrium electrostatic potential.
λ : channel-length modulation parameter.
γ : body-effect coefficient.

ELogic uses either a small-signal model or a line-thru-origin model for MOS transistors to linearize the device. The proper model to use depends on the situation and will be addressed later. The small-signal model and the line-thru-origin model are shown in Figure 2.5. The small-signal model represents the nonlinearity of the MOS transistor by an incremental conductance at the operating point and a current source in parallel(or a voltage source in series). This model is also referred to as the discretized model in the literature [24]. From Figure 2.5(a), the small-signal model is given by :



(a) Small-Signal Model



(b) Line-Thru-Origin Model

Fig. 2.5 (a) Small-Signal Model (b) Line-Thru-Origin Model

$$G_{SMALL} = \frac{\partial i}{\partial v}\Big|_{v = E_Q,\, i = J_Q} \tag{2.1}$$

$$I_Q = J_Q - G_{SMALL}\ E_Q \tag{2.2.a}$$

$$V_Q = -\, I_Q \,/\, G_{SMALL} \tag{2.2.b}$$

All equations necessary for getting a small-signal model for various terminal connections are derived from the Schichman-Hodges model equations and are shown in Table 2.2, where,

$$g_m = \frac{\partial i_d}{\partial V_g},\ g_b = \frac{\partial i_d}{\partial V_b},\ g_d = \frac{\partial i_d}{\partial V_d},\ g_s = \frac{\partial i_d}{\partial V_s}$$

Using the equations in Table 2.2, the incremental conductance of MOS transistors with any terminal connection can be obtained. For example, a small-signal model looking back into the pass transistor from the source is given by :

$$G_{sd} = \frac{\partial i_s}{\partial vs} = -\frac{\partial i_d}{\partial vs} = -g_s = g_m + g_d + g_b \tag{2.3}$$

$$I_Q = J_Q - GE_Q = is - G_{sd} V_{sd} = -i_d + (g_m + g_d + g_b) V_{ds} \tag{2.4}$$

The small-signal model looking back into the depletion load from the source is given by :

$$G_{sd} = \frac{\partial i_s}{\partial vg} + \frac{\partial i_s}{\partial vs} = -g_m - g_s = g_b + g_d \tag{2.5}$$

$$I_Q = J_Q - GE_Q = is - G_{sd} V_{sd} = -i_d + (g_d + g_b) V_{ds} \tag{2.6}$$

In the line-thru-origin model, the nonlinearity is modeled by a simple conductance between drain and source, and there is no current source (no voltage source) involved. From Figure 2.5(b), the line-thru-origin model is given by :

$$G_{LTO} = \frac{J_Q}{E_Q} \tag{2.7}$$

The conductance between drain and source is independent of terminal connections and the terminal for which the line-thru-origin model is evaluated.

$T(type)$	NMOS(1), PMOS(- 1)
K	$KP \frac{W}{L}$
ϕ	$-2\ T\ \phi_F > 0$
V_T	$V_{TO} + T\ \gamma(\sqrt{\phi + T\ V_{sb}} - \sqrt{\phi})$
V_{gsEff}	$T\ (Vgs - VT)$
IF ($V_{gsEff} < 0$)	OFF
ELSE	
V_{dsEff}	$\min(\ TV_{ds}\ ,\ V_{gsEff}\ )$
i_d	$T\ K\ (V_{gsEff} - \frac{V_{dsEff}}{2})\ V_{dsEff}\ (1 + T\lambda V_{ds})$

Table 2.1 Schichman-Hodges MOS Model Equations

$T(type)$	NMOS(1), PMOS(- 1)
i_d	$T\ K\ (V_{gsEff} - \frac{V_{dsEff}}{2})\ V_{dsEff}\ (1 + T\lambda V_{ds})$
g_m	$T\ K\ V_{dsEff}(1 + T\lambda V_{ds})$
g_b	$g_m \frac{\gamma}{2\sqrt{\phi + T\ V_{sb}}}$
g_d	$K(V_{gsEff} - V_{dsEff})(1 + T\lambda V_{ds}) + K(V_{gsEff} - \frac{V_{dsEff}}{2})V_{dsEff}\lambda$
g_s	$-(g_m + g_b + g_d)$

Table 2.2 Useful Equations for Small-Signal Models

tables, e.g.,

$$G = T_G \, (\, \overline{V}_{IN}) \tag{2.8.a}$$

$$I = T_I \, (\, \overline{V}_{IN}), \quad \textit{or } 0 \text{ for } \textit{Line-Thru-Origin Model} \tag{2.8.b}$$

where T_G and T_I are tables, which take input voltage states $\overline{V}_{IN}$ and provide a corresponding linearized MOS transistor model. This process is illustrated in Figure 2.6. Once the fabrication technology is determined, the preliminary I–G tables can be obtained experimentally or using the Schichman-Hodges equations. The preliminary I–G tables refer to tables which contain the small-signal model and the line-thru-origin model of all different model types with a normalized W/L ratio. These preliminary tables are stored in a technology library for the program. When a simulation starts, ELogic obtains the actual W/L ratios and different terminal connections for all different devices while reading a circuit description. Then ELogic generates the final I–G tables for each different model/element type in the circuit using the preliminary I–G tables. The final tables are used during the simulation. Using the table

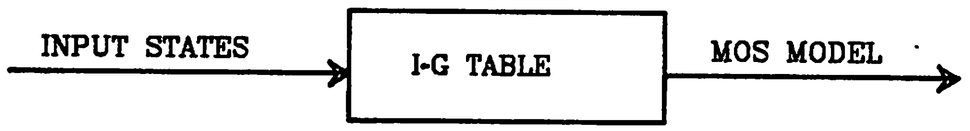
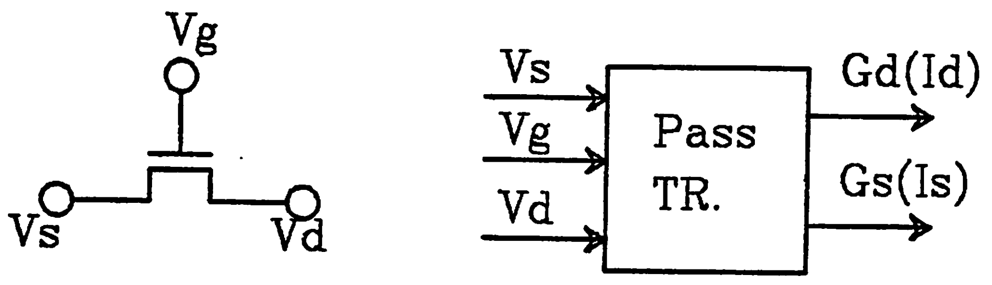
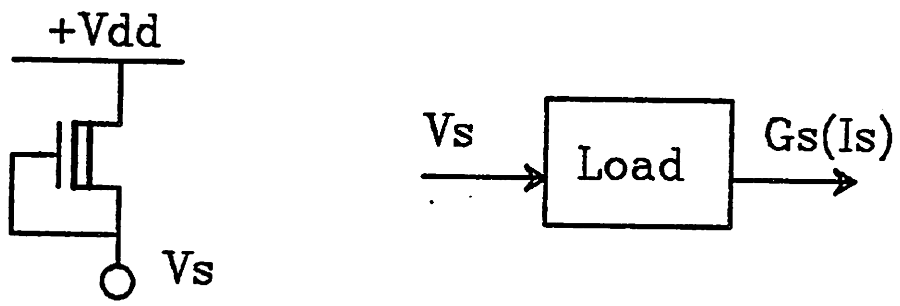


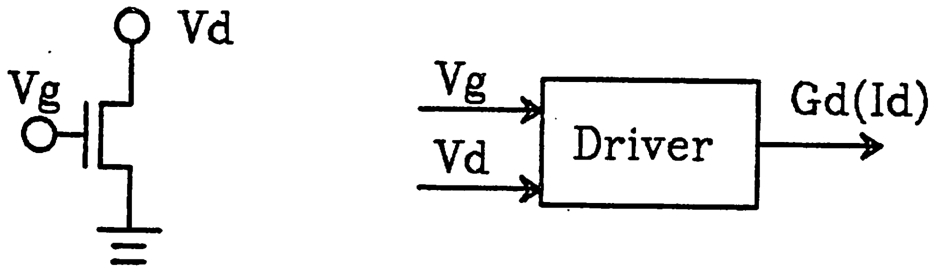
Fig. 2.6 ELogic MOS Transistor Models



(a) NMOS Pass Transistor



(b) NMOS Load



(c) NMOS Driver

Fig. 2.7 NMOS Transistors Table Model

preliminary I–G tables. The final tables are used during the simulation. Using the table look-up models, a simulation can be sped up. The process of generating the table look-up models efficiently, in terms of memory usage, is described in reference [25]. A detailed representation for various terminal connections of NMOS transistors are shown in Figure 2.7. For example, the NMOS pass transistor table(Figure 2.7(a)) takes, as input, the gate, source and drain voltage states, and provides conductance (and current) for the drain or source as output. The I–G tables for PMOS transistors work in a similar way.

2.6.2. ELogic Node Model

The other class of circuit element used in ELogic analysis is the *node* which consists of the node itself and a grounded capacitor, C_N (Figure 2.8). Here, the inputs to the node model are the Norton equivalent $\{I_j\ ,\ G_j\}$ pairs of the fanin elements and the fanin nodes. The outputs are the new node voltage, V_N, and the time point, T_N, at which V_N is achieved. The method that ELogic uses to calculate V_N and T_N depends on the particular algorithm in use, and will be described shortly.

2.6.3. Composite ELogic Model

An NMOS inverter and its composite ELogic model are shown in Figure 2.9. The signal flow graph between nodes and elements are illustrated in Figure 2.9(b), where a thin line represents a fanin node of an element and a thick line represents a fanin element of a node (the fanin node of a node is not shown here). The fanin nodes of element M_i is a set of nodes which directly influences the operating condition of the element M_i. Similarly, the fanin elements of node i are the elements which directly play some part in determining the voltage at node i. For example, input node A is a fanin node of driver $M1$ and $M1$ is a fanin element of output node B. If we look at the relationship between B and $M2$, not only is B a fanin node of $M2$ but also $M2$ is a fanin element of B. Therefore node B serves as an input node as well as an output node of the inverter. When ELogic calculates the next waveform frag-

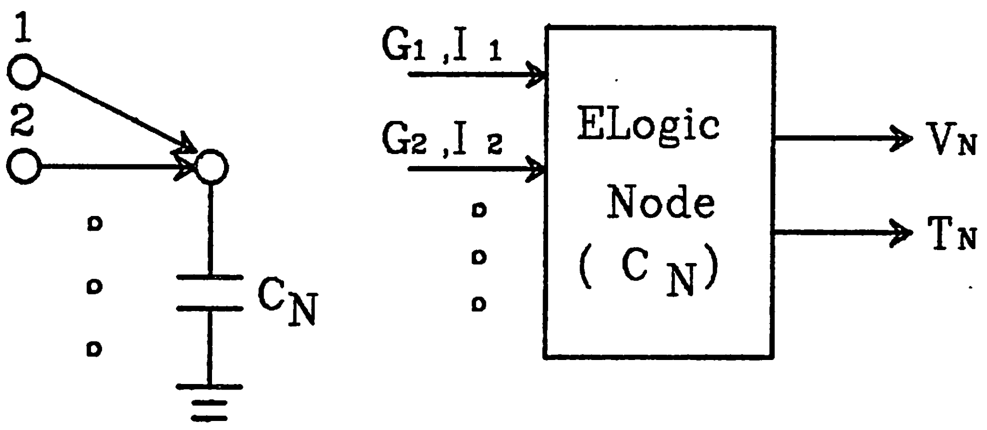
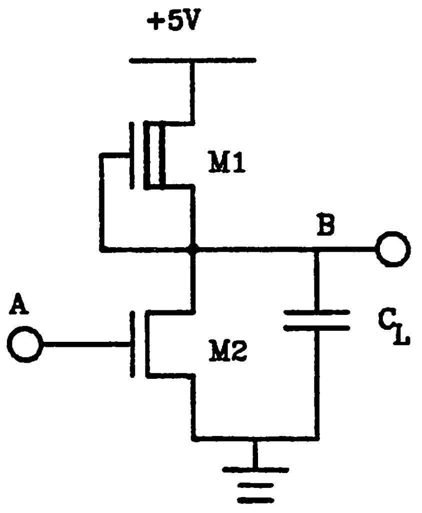


Fig. 2.8 Node Model for Basic ELogic



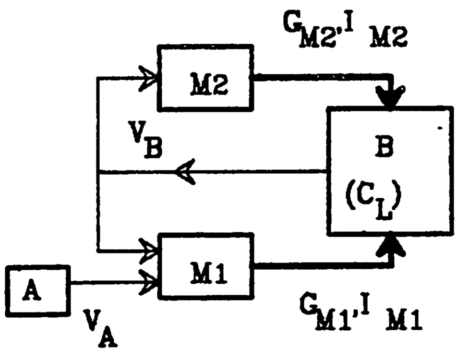


Fig. 2.9 (a) NMOS Inverter (b) Composite ELogic Model

ment, both A and B are taken into account. This is one of the reasons why ELogic is much more accurate than conventional multi-strength, multi-level switch-level simulators.

Assume that input node A achieves a new voltage state V_{t0} at t_0. Since A achieves a new state, its fanout node B must be processed at t_0. There are two fanin elements of B : $M1$ and $M2$. Therefore $M1$ and $M2$ obtain the discrete voltage states of its fanin nodes, A and B. Then $M1$ and $M2$ provide the Norton equivalent $\{I_j\ ,\ G_j\}$ pairs from I–G tables to node B. Using this information, B determines the next voltage state and calculates the time for that transition. When evaluating MOS transistors, I–G tables use discrete voltage states as input. Even though ELogic solves for the time at which the next discrete state is achieved, the terminal voltages of MOS transistors may be between the discrete states, and if so, it must be rounded off to the discrete state. In this case, the next voltage state to which the terminal voltage is moving is used as input to the I–G tables.

2.7. Event Scheduling and Canceling

The fanin nodes of node i are defined as a set of nodes whose voltage change directly influences the voltage change at node i, through one of the fanin elements connected to node i. Similarly, the fanout nodes of node i are those whose voltage is directly influenced by the voltage change at node i through one of fanout elements of node i. As an example, a MOS transistor which is modeled by Schichman-Hodges equations [23] and its associated fanin-fanout node-signal flow graph are shown in Figure 2.10. If a directed edge is connected from node i to j, node i is a fanin node of j and node j is a fanout node of i.

The ELogic implementation of an event-driven selective-trace can be described using the associated signal graph. The term event-driven selective-trace means that ELogic processes the node i at t_n only in the following two cases.

(1) When Node i achieves new voltage state at t_n(event-driven)
(2) When the fanin node of node i achieves new voltage state at t_n(selective-trace)

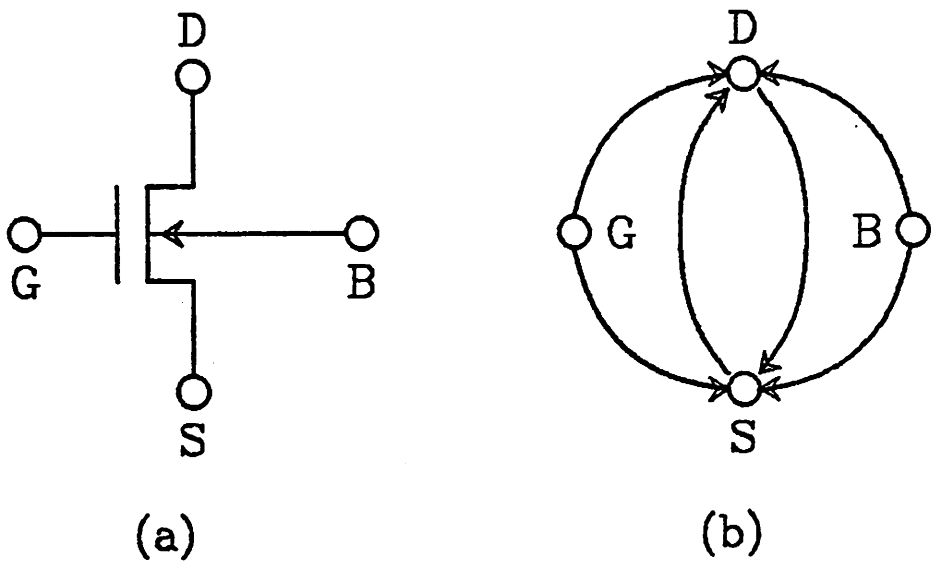


Fig. 2.10 (a) NMOS TR Modeled by Schichman-Hodges Equations
(b) Associated Node-Signal Flow Graph

Processing a node i at t_n involves the following steps : When ELogic-time reaches at t_n, ELogic obtains the state $S_i(t_n) = V_i$ at t_n (called updating the voltage at node i) and next voltage state $S_i(t_{n+1}) = V_{k+1}$. Then ELogic computes a corresponding time for the transition, h, and schedules node i in the time queue at future time point ($t_{n+1} = t_n + h$) so that node i can be processed when ELogic-time reaches t_{n+1}.

Consider the node-signal flow graph shown in Figure 2.11. If the voltage at node A moves through its discrete set of voltages, the time points at which new values of node B and C will be attained are computed. Assume that node A achieves its new state. Then nodes A is re-processed and the adjustment for fanout nodes B and C is done. If node B had been scheduled in the time queue already (this happens when input transition is fast), the pending event is canceled and a new voltage state and a new transition time are computed. Node B is

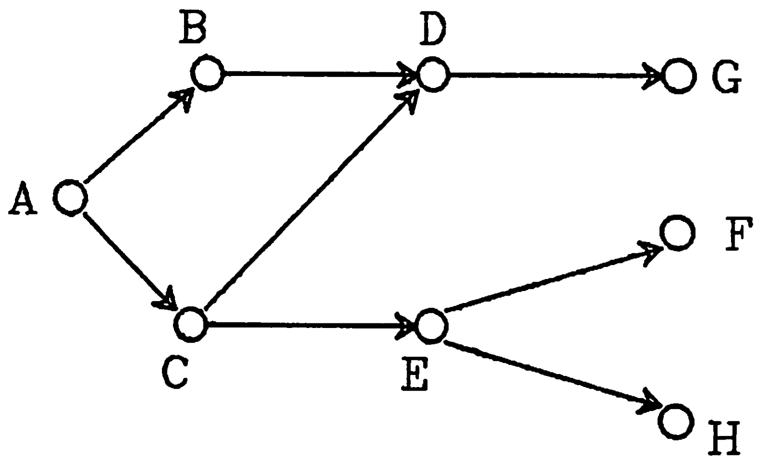


Fig. 2.11 Example of Node-Signal Flow Graph

then re-scheduled at the new time point. This procedure is called *fanout adjustment.* If the other fanout node, C, were not in time queue (e.g., node C is latent), it may or may not be scheduled latent, depending on the electrical properties of the circuit being analyzed. This procedure is called ***activating latent fanout nodes.***

2.8. Relaxation-Based Electrical Simulation

2.8.1. Relaxation-Based Electrical Simulation

Consider a set of linear equations of the form :

$$A\,x = b, \qquad where\ A \in \mathbf{R}^{n \times n}\ ,\ x,b \in \mathbf{R}^{n} \tag{2.9}$$

Standard circuit simulators such as SPICE2 [3] or ASTAP [4] use direct methods to solve the above equations, such as Gaussian Elimination or LU Decomposition. However if Eqn. (2.9) is sparse and matrix A has a diagonal dominance property, relaxation methods such as Gauss-

Jacobi or Gauss-Seidel algorithms can be more efficient in terms of CPU-time. Both Gauss-Jacobi and Gauss-Seidel generate a sequence of approximate solutions $\{x^k\}$ and the iterations continue until $\{x^k\}$ converges to $A^{-1}b$. Gauss-Jacobi differs from Gauss-Seidel in that it does not use the most recent information when computing x_i^{k+1} while Gauss-Seidel does. Further details about applying direct and indirect methods to the solutions of a set of linear equations may be found in references [24,26,27,28].

Relaxation methods decouple a system of equations and solve them using an iterative method. Since it involves each equation separately, both temporal latency and spatial latency can be exploited. The terms temporal latency and spatial latency are used to mean the latency in a waveform over a time period and the latency in the circuit at a given time point, respectively. A family of the relaxation-based electrical simulators are compared to standard electrical simulation in reference [29].

Timing simulation is an early form of relaxation-based circuit simulation which was introduced in the mid-seventies [30,31]. Even though timing simulation programs use relaxation methods to solve the set of circuit equations, none of these programs carries the iterations to convergence. Each equation is solved only once at each time point to reduce cost, while one or more Newton-Raphson iterations may be used. The accuracy of timing simulation is maintained using a small time step for the whole simulation period. In fact, timing simulators have been successfully used for constrained IC design methods such as gate array or standard cell. However the selection of an appropriate step size is difficult and it may be limited by stability considerations. Therefore it sometimes has difficulties in solving custom-designed circuits which contain strong coupling. Thus, the Iterated Timing Analysis and Waveform Relaxation have been developed.

In Iterated Timing Analysis (ITA) [2,13,5], nonlinear relaxation is used to solve a system of nonlinear equations obtained by applying numerical integration method to capacitors and inductors in the network. ITA is similar to timing simulation except that it carries the outer relaxation loop to convergence. ITA exploits waveform latency by processing only the nodes which are changing at each time point. The remaining nodes are updated using their values at the previous time point. However, only one Newton-Raphson iteration is performed per equation for each Gauss-Seidel iteration.

Waveform Relaxation (WR) [6,32,33] is another form of relaxation-based electrical simulation. In this method, the relaxation is applied at the differential equation level. The difference of the basic WR from ITA is that WR solves for one variable (waveform) of each equation for the entire simulation period[1] while fixing the other variables in the equation, rather than solving for all variables at a given time point as ITA does. WR algorithm has also captured considerable attention due to its favorable numerical properties and has been successfully applied to the analysis of large MOS digital circuits. A modified version of WR algorithm has been implemented in program RELAX2 [32].

2.8.2. Convergence of Relaxation-Based Methods

The convergence properties of relaxation methods can be understood by looking at the convergence properties of the linear relaxation methods which solves Eqn. (2.9). The following is a well known theorem on the convergence property of linear relaxation methods : [28].

Theorem 2.1 [G. Golub and C. Van Loan]

Let A be split into $L + D + U$, where $L \in \mathbb{R}^{n\times n}$ is strictly lower triangular, $D \in \mathbb{R}^{n\times n}$ is diagonal, and $U \in \mathbb{R}^{n\times n}$ is strictly upper triangular. Let $N \in \mathbb{R}^{n\times n}$, $N = -(L + U)$. It is known that if D is nonsingular and the spectral radius of $D^{-1}N$ satisfies $\rho(D^{-1}N) < 1$, then the iterates $\{x^k\}$ converges to $A^{-1}b$.

One condition that guarantees $\rho(D^{-1}N) < 1$ is strict diagonal dominance of A. Clearly, the

[1] WR algorithm may break the simulation period into pieces, called windows, so that the number of relaxation iterations required to achieve convergence can be reduced. This is called windowed WR [32].

more dominant the diagonal of A, the more rapid will be the convergence. All relaxation-based electrical simulators assume a grounded capacitor at each node. Since the conductance of the grounded capacitors appear only in the diagonal entries of the matrix A after an integration method is applied, it helps these methods obtain convergence. If there is strong coupling in the circuit, the relaxation-based methods still obtain convergence but the convergence speed becomes very slow. In this case, both ITA and WR requires more iterations to get convergence. One of the techniques to solve this convergence speed problem of relaxation-based methods in the presence of strong coupling is by partitioning the system variables. That is, the program or the user partitions the circuit into loosely coupled sub-circuits by grouping tightly coupled nodes together. The sub-circuits are solved using direct methods and relaxation is applied between sub-circuits [12].

2.9. ELogic Algorithms

Three ELogic algorithms, ELogic-1, ELogic-2 and ELogic-3 respectively, are proposed and described in the remainder of this chapter. The ELogic-1 (basic ELogic) is the simplest one and it is used with relatively small voltage steps. Floating capacitors are not allowed in the circuit description. ELogic-2 is more complex than ELogic-1 but simpler than ELogic-3. No floating capacitor is allowed either but larger voltage steps may be used than ELogic-1. ELogic-3 is the most complex and floating capacitors are allowed in the circuit description. All ELogic algorithms are similar to timing simulations since they do not carry the iterations to convergence and, do not iterate between nodes while processing. The main difference between ELogic-1 and ELogic-2 (ELogic-3) is the way they model the grounded capacitor as will be explained shortly.

Consider the relaxation process for each equation at each iteration, after decoupling the system. At each iteration of the Gauss-Seidel algorithm which solves Eqn. (2.9), x_i^{k+1} is computed according to Eqn. (2.10), where x_i^{k+1} is the $(k+1)$th approximation of x_i.

$$x_i^{k+1} = \frac{(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k)}{a_{ii}} \tag{2.10}$$

The Gauss-Jacobi iteration is obtained by replacing x_j^{k+1} with x_j^k in the above equation. At each iteration, all values of x, except x_i, are fixed values in both algorithms. This means that when the relaxation method solves for a voltage at node i at each iteration, it regards the fanin nodes of i as ideal constant voltage sources with no resistance. Therefore all nodes except the fanin nodes play no role in determining the voltage at node i at each iteration. Hence such nodes are simply ignored. After solving for the node-voltage at i, the next node in sequence is processed in the same way. This procedure is repeated until all node-voltages converge. This is a physical interpretation of the way that the relaxation-based method solves the circuit at each time point.

Similarly, when ELogic algorithms process a node i, they decouple a sub-circuit, consisting of only node i and its fanin nodes, from a given circuit. Then, ignoring the remainder of the circuit, the transition time for the node i to achieve next state is calculated from the information associated with the sub-circuit.

2.9.1. Modeling of the Grounded Capacitor

Four ways of discretizing the grounded capacitor are as follows [24, 34].

(1) Constant voltage source

(2) Discrete model by Forward Euler method

(3) Discrete model by Backward Euler method

(4) Discrete model by Trapezoidal method

The first model is used in most timing simulators and for each iteration of relaxation-based electrical simulators, as explained before. Let V_n be the voltage across the capacitor being modeled and i_n be the current through the capacitor, at nth time point. Then the Forward Euler method is :

$$V_{n+1} = V_n + h\dot{V}_n \tag{2.11}$$

The Backward Euler method is :

$$V_{n+1} = V_n + h\ \dot{V}_{n+1} = V_n + h\ \frac{i_{n+1}}{C} = E_{BE} + R_{BE}i_{n+1} \tag{2.12}$$

The Trapezoidal method to model the grounded capacitors is :

$$\begin{aligned} V_{n+1} &= V_n + \frac{h}{2}(\dot{V}_n + \dot{V}_{n+1}) = V_n + \frac{h}{2C}(i_n + i_{n+1}) \\ &= (V_n + \frac{hi_n}{2C}) + \frac{h}{2C}i_{n+1} = E_{TZ} + R_{TZ}i_{n+1} \end{aligned} \tag{2.13}$$

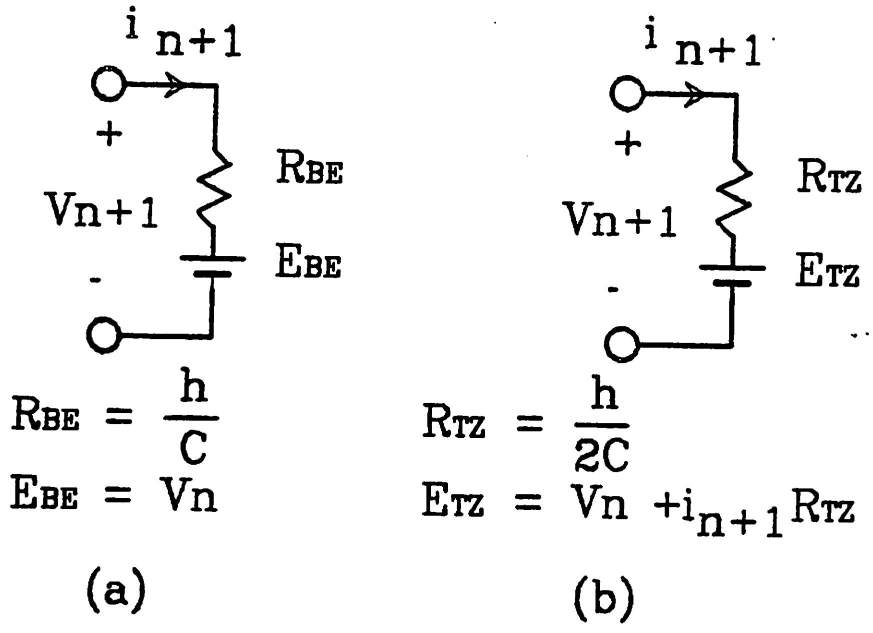


Fig. 2.12 Discrete model of capacitor using (a) Backward Euler (b) Trapezoidal

The discrete circuit models of the grounded capacitor, using the Backward Euler and the Trapezoidal integration method, are shown in Figure 2.12. Refer to references [24, 35] for their numerical properties.

The basic ELogic algorithm (ELogic-1) uses the constant voltage source model for the grounded capacitor at each fanin node. The voltage value is equal to the voltage across the grounded capacitor at the corresponding fanin node, at the given time point. However, it uses the Forward Euler method for the grounded capacitor at the node under analysis. ELogic-2 and ELogic-3 use both the constant voltage source model and the discrete model using the Trapezoidal integration method.

2.9.2. Basic ELogic Algorithm

Consider a circuit fragment shown in Figure 2.13(a), where the node A is being processed. Since the fanin nodes of node A are regarded as being connected by constant voltage sources; the circuit can be simplified to Figure 2.13(b). The norton equivalent of its fanin nodes is given as follows :

$$G_{NORTON} = \sum_{i=1}^{n} G_i \tag{2.14.a}$$

$$I_{NORTON} = \sum_{i=1}^{n} (V_i - V_A) G_i \tag{2.14.b}$$

The transition time, h, is calculated using a Forward Euler model for the grounded capacitor of node A, C_A.

$$V_{n+1} = V_n + h\dot{V}_n \tag{2.15.a}$$

$$h = \frac{V_{n+1} - V_n}{\dot{V}_n} = \frac{(V_{n+1} - V_n)C_A}{i_n} = \frac{(V_{n+1} - V_n)C_A}{I_{NORTON} - V_n G_{NORTON}} \tag{2.15.b}$$

Assume $V_{n+1} > V_n$, e.g., V_A is going up. Let the threshold voltage be, V_{THRESH}, be defined as the midpoint value of two adjacent states :

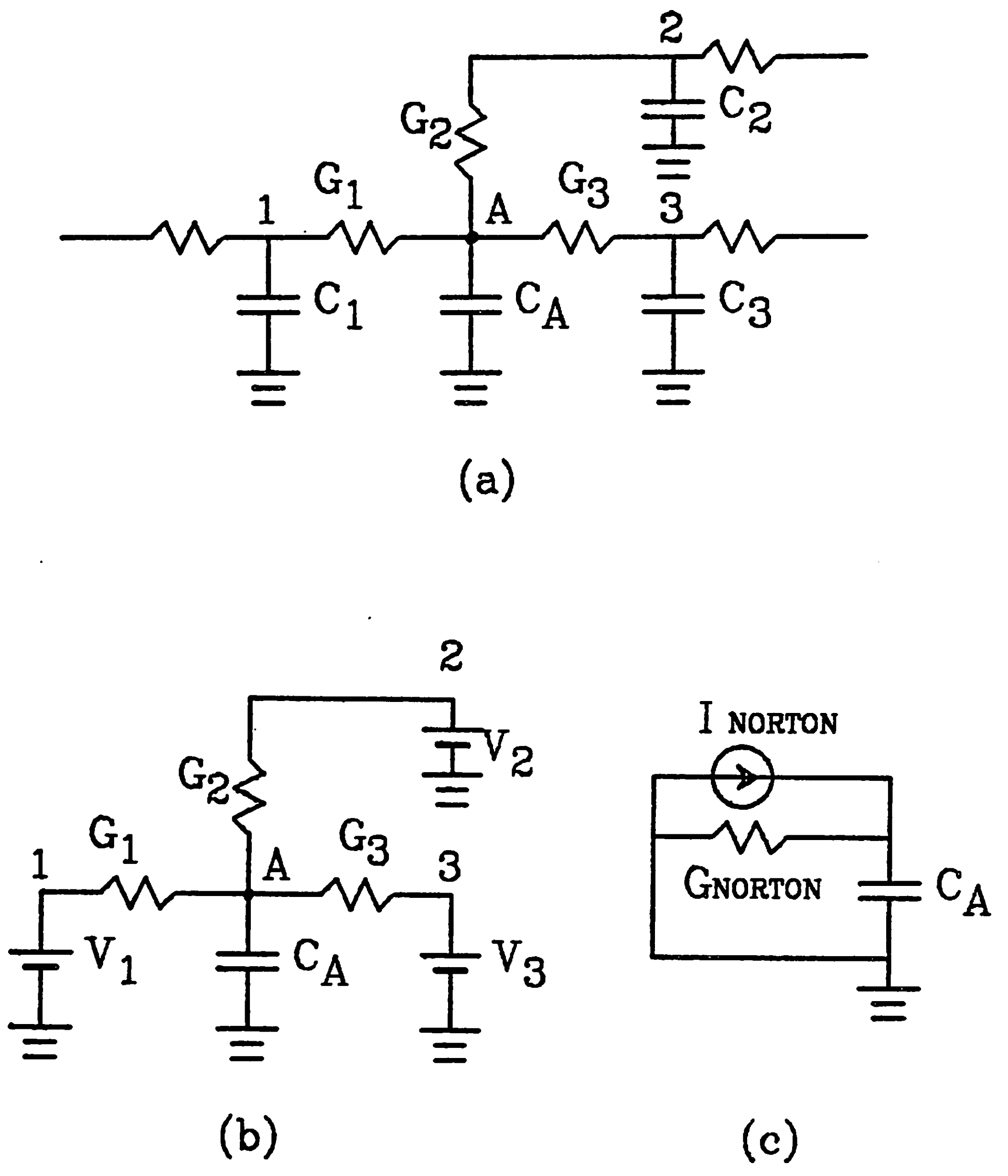


Fig. 2.13 (a) Circuit Fragment (b) Simplified Circuit (c) Norton Equivalent

$$V_{THRESH} = \frac{V_n + V_{n+1}}{2} \tag{2.16}$$

When the Thevenin equivalent voltage

$$V_{THEV} = I_{NORTON} / G_{NORTON} \tag{2.17}$$

is larger than V_{THRESH}, Eqn. (2.15.b) is used to calculate the transition time. If V_{THEV} is smaller than V_{THRESH}, there will be no transfer and node A is not scheduled.

The flow of the basic ELogic algorithm is shown below.

```
/* main program */

Basic_ELogic()
{
  setup();
  proc();
}

/* read, construct data structures and initialization */

setup()
{
  readin();

/* process all nodes at t=0 */

  forall (nodes j in the circuit) {
    if (PLOT)    print(j);
    get next voltage state;

/* solve for next time point */

    if (get_time(j) is not FALSE)  schedule(j at next time point);
    else    do_nothing;    /* j is inactive */
  }
}

proc()
{
  while ( time queue is not empty) {
    get next node, j, from queue;
    update ELogic_time;
    if (PLOT)    print(j);

/* Try self-scheduling */

    get next voltage state for j;
    if (get_time(j) is not FALSE)  schedule(j at next time point);
```

```
      else    do_nothing;    /* $j$ is inactive */

  /* Fanout node adjustment */

      forall (fanout nodes $k$ of $j$) {
        if ($k$ is in the queue) {
          delete $k$ from queue;
          if (get_time($k$) is not FALSE)
             schedule($k$ at next time point);
          else
             do_nothing;    /* $k$ became inactive */
        }

  /* Try activating inactive fanout nodes */

        else {
          if (get_time($k$) is not FALSE)
             schedule($k$ at next time point);
          else    do_nothing;    /* $k$ remains inactive */
        }
      }
    }
  }

  /* compute the transition time for $j$ to achieve next state */

  get_time($j$)
  {
    forall(fanin nodes $i$ of node $j$) {
      evaluate the fanin elements between $i$ and $j$ by table;
      model node, $i$, by constant voltage source;
      update $G_{NORTON}$, $I_{NORTON}$;
    }
    compute a transition time; /* FE */
  }
```

2.10. Strong Coupling in ELogic

Strong coupling in a circuit slows the convergence speed of relaxation-based methods, as described earlier. Since the basic ELogic also belongs to the family of relaxation-based methods, strong coupling deserves special attention. Consider a test circuit, shown in Figure 2.14(a), where node A and B are strongly coupled. Assume node A is being processed. Let $V_A(0) = V_B(0) = 0$ V. Using the basic ELogic algorithm, a fanin node B is virtually grounded, since the voltage across its grounded capacitor C_B is zero(Figure 2.14(b)). The Norton Equivalent pair for node A obtained from its fanin elements is given as follows.

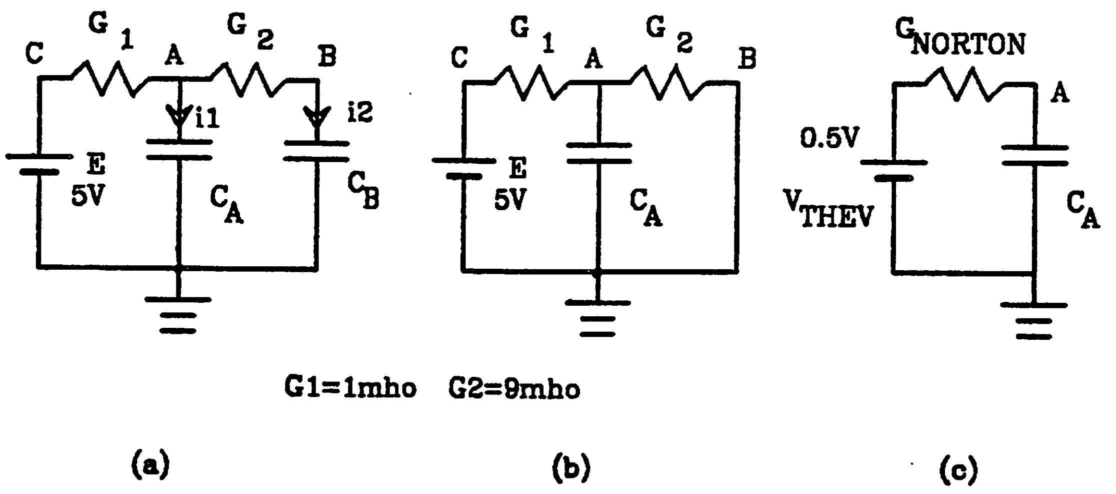


Fig. 2.14 (a)Example of Strong Coupling (b) Equivalent Circuit seen from Node A,using the Basic ELogic (c) Final Equivalent Circuit

$$I_{NORTON} = 5\ G_1 = 5 \times 1 = 5\ (A) \qquad (2.18.a)$$

$$G_{NORTON} = G_1 + G_2 = 1 + 9 = 10\ (mho) \qquad (2.18.b)$$

Therefore the Thevenin equivalent voltage (Figure 2.13(c)),

$$V_{THEV} = \frac{5\ G_1}{G_1 + G_2} = 0.5\ V \qquad (2.18.c)$$

From Figure 2.14(c), if the voltage step is larger than 1V ($2V_{THEV}$), basic ELogic claims V_A can not transfer to next state and V_A, and consequently V_B, remains at same voltage which is zero V. The maximum voltage step which can be used in the basic ELogic depends on the ratio of G_1 and G_2. While strong coupling slows the convergence speed of ITA and WR, it results in a transition time error in the case of basic ELogic, since it does not perform iterations to convergence. The voltage step of 0.1 V to 0.25 V works well for practical NMOS circuits.

Consider the strong coupling example again, using the Backward Euler integration method for grounded capacitors. Assume that node A is being processed as in previous example and let $V_A(0) = V_B(0) = 0$ V. Figure 2.15(a) is the equivalent circuit of Figure 2.14, using the Backward Euler integration method. In Figure 2.15(a), node B is no longer grounded and it has a conductance to ground, whose value is $f(h)$. Since $f(h)$ goes to zero(open) as h goes to infinity and the transition time h is the unknown which we are going to compute, node A can transfer to next state no matter how large the voltage step is in the presence of the strong coupling in circuits. Therefore the Backward Euler integration method can be successfully applied to the grounded capacitor in this example. Similarly, the Tra-

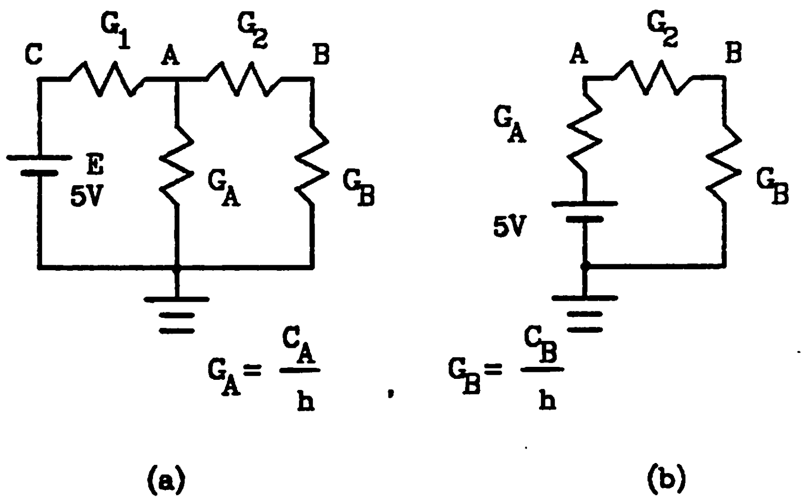


Fig. 2.15 Test Circuit after Applying BE. Solving for:
(a) Node A (b) Node B

pezoidal integration method also can be used to model the grounded capacitor in this example.

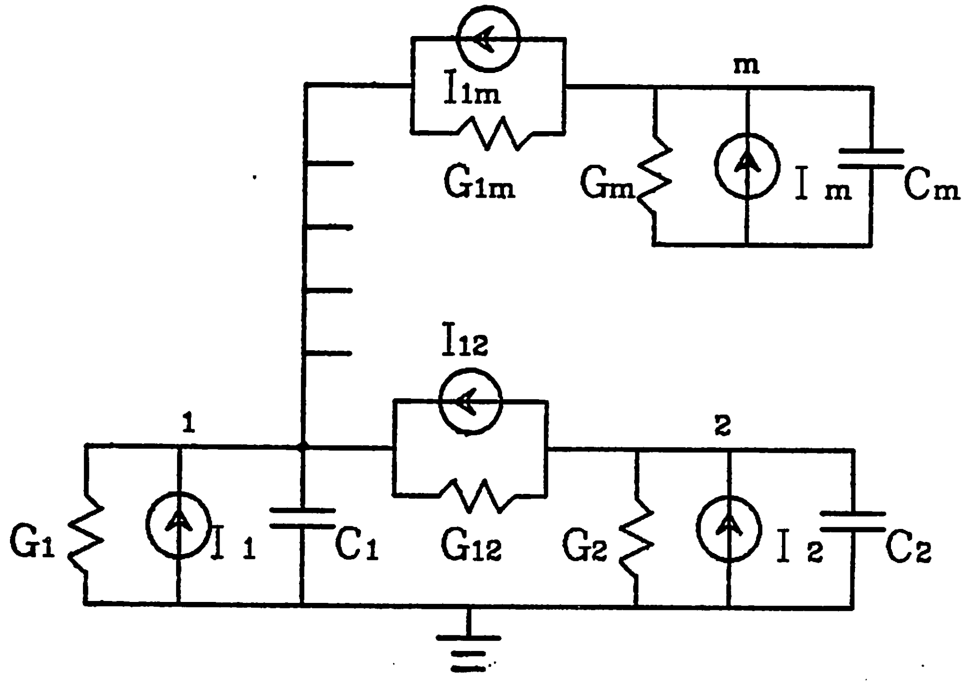
Now consider the test circuit shown in Figure 2.14(a) once more. Assume that node B is being processed at this time and let $V_A(0) = 5$ V and $V_B(0) = 0$ V. When processing node B, ELogic ignores node C as mentioned previously. The equivalent circuit when processing node B is shown in Figure 2.15(b) after applying the Backward Euler method to C_A. Therefore $V_B(h)$ is given as follows, where the resistance $R_2 = \frac{1}{G_2}$.

$$V_B(h) = \frac{5R_B}{R_A + R_2 + R_B} \tag{2.19}$$

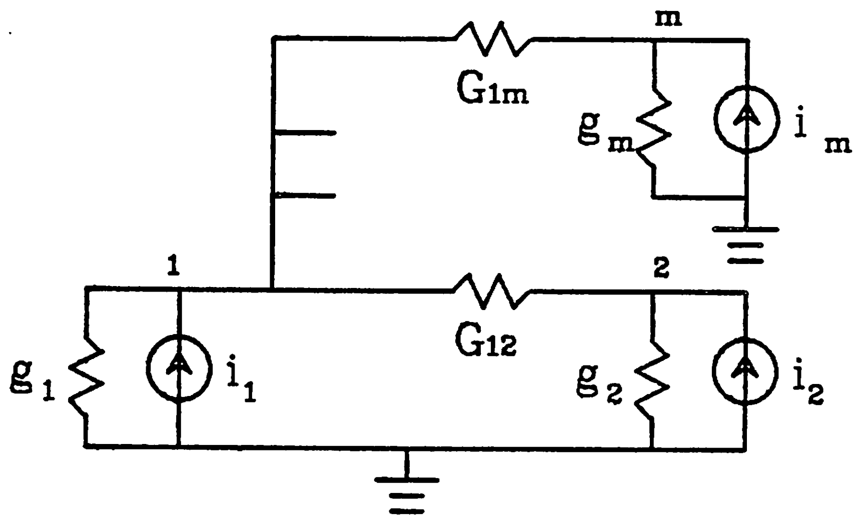
Let R_A be equal to R_B ($C_A = C_B$). Then $V_B(h)$ always smaller than 2.5V regardless of the value of h. If the voltage step is 5V, ELogic will claim that node B can never reach 5V unless $R_2 = 0$. Even though the voltage step is chosen as other than 5V, $V_B(h)$ still can not reach 5V and it remains at one step below 5V. The solution for this is to use both a constant voltage source model and either one of the Backward Euler or the Trapezoidal method together. In other words, if the fanin node helps the transition of the node under analysis, it is modeled as a constant voltage source. Otherwise, either the Backward Euler or Trapezoidal method is used.

2.10.1. Discretized Newton-Raphson Method

Consider the circuit shown in Figure 2.16(a), where node 1 is being processed by ELogic-2. Figure 2.16(b) is the simplified circuit which ELogic-2 uses when it solves for node 1. This is obtained from Figure 2.16(a) by ignoring all nodes except node 1 and its fanin nodes. For the sake of simplicity, the Backward Euler method is used to model all the grounded capacitors. ELogic-2 and ELogic-3 use the Trapezoidal method and will be discussed later. The fanin nodes which are modeled by the constant voltage source may be included in I_1 and G_1, using their Norton equivalent. In Figure 2.16(b),



(a)



(b)

Fig. 2.16 (a) Circuit Example (b) Simplified Circuit

$$i_1 = I_1 + \sum_{j=2}^{m} I_{1j} + V_1^n \frac{C_1}{h} \tag{2.20.a}$$

$$i_j = I_j - I_{1j} + V_j^n \frac{C_j}{h}, \qquad \text{for } j = 2,...,m \tag{2.20.b}$$

$$g_j = G_j + \frac{C_j}{h}, \qquad \text{for } j = 1,...,m \tag{2.20.c}$$

, where V_j^n is the voltage at node j at the nth time point. The nodal analysis equations to solve Figure 2.16(b) at $(n+1)$th time point are given as follows.

For node 1,

$$(g_1 + \sum_{j=2}^{m} G_{1j}) V_1^{n+1} - \sum_{j=2}^{m} (G_{1j} V_j^{n+1}) = i_1 \tag{2.21.a}$$

For nodes j=2,3,...,(m-1),m,

$$-G_{1j} V_1^{n+1} + (g_j + G_{1j} V_j^{n+1}) = i_j \tag{2.21.b}$$

From Eqn(2.21.b),

$$G_{1j} V_j^{n+1} = \frac{G_{1j} i_j + G_{1j}^2 V_1^{n+1}}{g_j + G_{1j}} \tag{2.21.c}$$

Substituting Eqn. (2.20) and (2.21.c) into Eqn. (2.21.a) and rearranging, we obtain

$$\begin{aligned} &\frac{C_1}{h}(V_1^{n+1} - V_1^n) + (G1 V_1^{n+1} - I_1 - \sum_{j=2}^{m} I_{1j}) \\ &+ \sum_{j=2}^{m} [G_{1j} \frac{C_j(V_1^{n+1} - V_j^n) + h(G_j V_1^{n+1} - (I_j - I_{1j}))}{C_j + h\,(G_j + G_{1j})}] = 0 \end{aligned} \tag{2.22}$$

Since

$$\sum_{j=2}^{m} \frac{1}{A_j + B_j} = \frac{\sum_{j=2}^{m} [\prod_{j=2, i \neq j}^{m} (A_i + B_i)]}{\prod_{j=2}^{m} (A_j + B_j)} \quad , \tag{2.23}$$

Eqn. (2.24) given below is an mth order polynomial obtained from Eqn. (2.22) after rearrangement.

$$
\begin{aligned}
f(h) = & \sum_{j=2}^{m}[G_{1j}(G_j V_1^{n+1} - (I_j - I_{1j}))\, h^2 \prod_{j=2, i\neq j}^{m} (C_j + h(G_j + G_{1j}))] \\
& + (G_1 V_1^{n+1} - I_1 - \sum_{j=2}^{m} I_{1j})\, h \prod_{j=2}^{m}(C_j + h(G_j + G_{1j})) \\
& + \sum_{j=2}^{m}[G_{1j} C_j (V_1^{n+1} - V_j^n)\, h \prod_{j=2, i\neq j}^{m} (C_i + h(G_i + G_{1i}))] \\
& + C_1(V_1^{n+1} - V_1^n)\prod_{j=2}^{m}(C_j + h(G_j + G_{1j})) = 0
\end{aligned}
\tag{2.24}
$$

In the above equation, C_j and $h(G_j + G_{1j})$ are comparable in the order of magnitude in practical circuits and neither one may be ignored. Note that the fanin node which is modeled by the constant voltage source does not contribute to the order of the Eqn. (2.24) and m is the number of the nodes which are modeled by Backward Euler, including the grounded capacitor at the node being processed. If the grounded capacitor at the node being processed is the only capacitor modeled by Backward Euler method, the transition time can be obtained by ELogic-1. If Eqn. (2.24) is a quadratic or a cubic equation, the transition time, h, can be computed easily using the known formula for the roots. However, if the order of the Eqn. (2.24) is higher than 3, the Newton-Raphson iteration method is necessary [24, 26]. The Newton-Raphson method for one equation in one unknown is shown in Eqn. (2.25).

$$
h^{k+1} = h^k - \frac{f(h^k)}{f'(h^k)}
\tag{2.25}
$$

,where h^k is the kth iteration value of h. It is well known that, if the initial guess h^0 is sufficiently close to a correct solution h^* of Eqn. (2.24), then the Newton-Raphson algorithm will always converge to h^* and its rate of convergence is asymptotically quadratic. A geometrical interpretation of the algorithm may be found in reference [24]. However, since it is difficult to obtain $f'(h)$ form Eqn. (2.24) explicitly, the Newton-Raphson algorithm can be approximated by the discretized one as follows :

$$\begin{aligned} h^{k+1} = h^k - \frac{f(h^k)}{f'(h^k)} &\approx h^k - \frac{\Delta h \ f(h^k)}{f(h^k + \frac{\Delta h}{2}) - f(h^k - \frac{\Delta h}{2})} \\ &\approx h^k - \frac{\Delta h \ f(h^k)}{f(h^k) - f(h^k - \Delta h)} \end{aligned} \tag{2.26}$$

Eqn. (2.26) is used to obtain the transition time, h, when the order of Eqn. (2.24) is higher than 3.

2.10.2. ELogic-2 Algorithm

Eqn. (2.26) requires the evaluation of $f(h)$ at two different values of h for each Newton-Raphson iteration. Either Backward Euler method or Trapezoidal method can be used to model the grounded capacitor at strongly coupled fanin nodes so that they can be solved together with node j which is being solved. But ELogic-2 (ELogic-3) uses Trapezoidal integration which is a second-order method rather than Backward Euler which is a first-order method, for better accuracy.

The following algorithm is used in ELogic-2 to model the grounded capacitor and to compute the transition time. The other algorithms are the same as those of ELogic-1. Note that if all fanin nodes are modeled by a constant voltage source, ELogic-1 can be used to solve for the transition time.

```
get_time($j$) {
   linearized the circuit for ELogic-2;
   decide the direction of voltage change at $j$;
   modelC_2 ($j$); /* model C */

/* calculate the transition time */
   if (# of fanin nodes modeled by TZ = 0)
      use ELogic-1 algorithm;
   else if ( # of fanin nodes modeled by TZ $\leq$ 2 )
      use the root-formula to solve quadratic(cubic) equation;
   else
      use the discretized Newton-Raphson;
}

modelC_2($j$) {
   forall(fanin nodes $i$ of $j$) {
   if ($i$ helps the voltage change at $j$)
```

```
            model by a constant Vtg Src;
        else
            model by Trapezoidal;
        }
    }
```

2.11. Bi-Directional Pass Transistor

Since ELogic does not have uni-directional pass transistor models, the pass transistor always implies a bi-directional pass transistor throughout the report. The pass transistor can introduce strong coupling into a circuit. Consider a pass transistor which follows an NMOS inverter. Usually, the load transistor of an NMOS inverter has higher resistance than the driver so that V_{OL} can be kept low. However the size of the pass transistor is comparable to that of the driver. If the driver of the NMOS inverter is off, the pass transistor couples its drain and source nodes much more tightly than the depletion load transistor does. If a small voltage step (in practice, not larger than 0.25 V) is defined for ELogic, then ELogic-1 is sufficient to solve circuits with pass transistors. However, if a large voltage step is desired, ELogic-2 must be used.

Recall the small-signal model of Figure 2.5(a) for pass transistors. Since the model contains either a dc current source (I_Q) or a dc voltage source (V_Q), this dc source sometimes helps ELogic to determine correctly whether or not the node under analysis can transfer to next state. However, occasionally, it may prevent a transfer when one should be made. An example of the latter case is illustrated in Figure 2.17(a), where a capacitor C_1 is charged through a pass transistor $M1$. The small-signal model for the source of M1 is shown in Figure 2.17(b). The equivalent circuit using the small-signal model is also shown in Figure 2.17(c). Since the dc voltage source of the small-signal model is negative ($V_Q < 0$), it makes the Thevenin equivalent voltage, V_{THEV} of Figure 2.17(c), lower than 5V ($V_{THEV} < 5$). Let $V_S(0)$ be 0 V. Then, if the voltage step is larger than $(2V_{THEV})$, ELogic incorrectly determines that node S can not transfer. If ELogic performs ITA-like iterations, V_S will be updated to a new voltage value and the pass transistor will be re-evaluated. Then after some iterations,

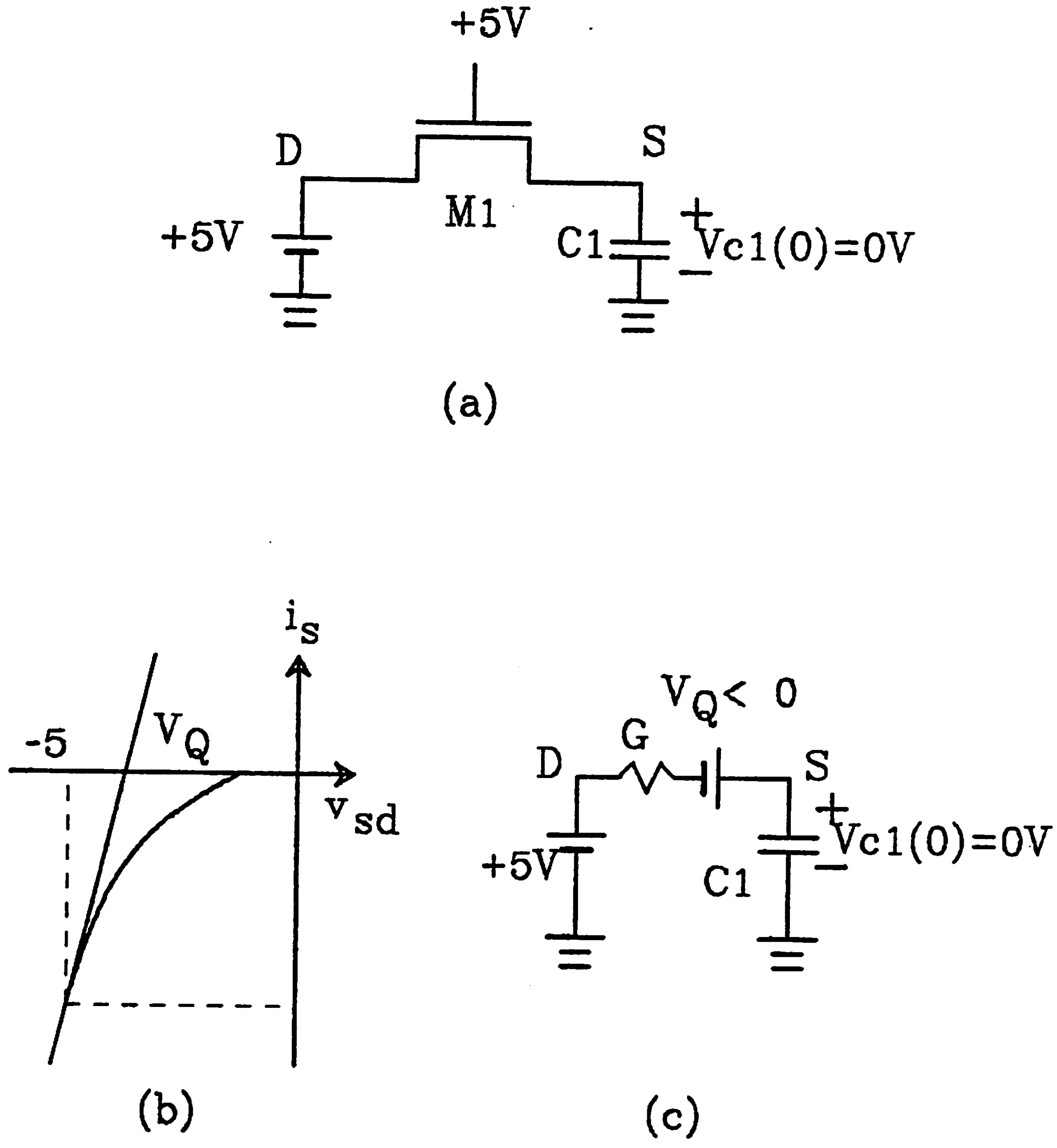


Fig. 2.17 (a) Pass TR Example (b) Small-signal Model for the Source of M1
(c) Equivalent Circuit

ELogic will determine that node S will transfer to next state. However ELogic belongs to the class of timing simulators and does not allow iterations.

The effect of the dc voltage source of the small-signal model is dependent of the dc parameters of MOS transistors and the way that terminals are connected to each other. The effect is summarized in Table 2.3, for various terminal connections for NMOS and PMOS, where *HELP* means it helps and *HINDER* means it hinders ELogic from making a correct decision on whether or not a node can transfer to next state.

As indicated in Table 2.3, if a large voltage step is desired in ELogic, the line-thru-origin model is more adequate than the small-signal model for the pass transistors. Both models can be used for NMOS driver, NMOS load and PMOS driver regardless of the value of the voltage step. However, since the small-signal model provides a more accurate waveform than the line-thru-origin model, it is better to use it for those transistors. ELogic-2 provides both models so that the user can choose the appropriate one, depending on the value of the voltage step and the terminal connections.

2.12. Floating Capacitor

A ***floating capacitor*** is defined as a capacitor such that neither of its terminals are connected to a reference node. It is only allowed in the circuit description of ELogic-3. The

MOS Type	Terminal Connection	State Change	Evaluated at	Behavior
NMOS	Pass TR	Up	Source	HINDER
	Pass TR	Down	Drain	HELP
	Driver	Down	Drain	HELP
	G-D tied	Down	Drain	HINDER
Load (NMOS)	G-S tied	Up	Source	HELP
PMOS	Pass TR	Down	Source	HINDER
	Pass TR	Up	Drain	HELP
	G-S tied	Up	Drain	HINDER

Table 2.3 Effect of Small-Signal Models in ELogic State Transition

floating capacitor can also create strong coupling between two nodes. When the nodal analysis equations of Eqn. (2.9) are formulated, the conductance which represents a floating capacitor appears as an off-diagonal element of A, as well as a diagonal element of A. As a result, a floating capacitor reduces the diagonal dominance of A, while a grounded capacitor improves it. Therefore floating capacitors have limited the application of timing simulators. In early timing simulators, floating capacitors were not allowed in the circuit description. Instead, the user approximated the effect of a floating capacitor by altering the values of the grounded capacitors at the appropriate nodes [30]. Recently, a number of studies have been made to allow floating capacitors in timing simulations, for example, the symmetric displacement technique [36] and the Implicit-Implicit-Explicit method [37,38].

A floating capacitor in ELogic adds some complexity to the algorithm. One example which requires special attention in calculating a transition time is illustrated in Figure 2.18. In this circuit, the voltage change at node 1 directly affects the voltage at node 2 without any time delay, according to the following equation.

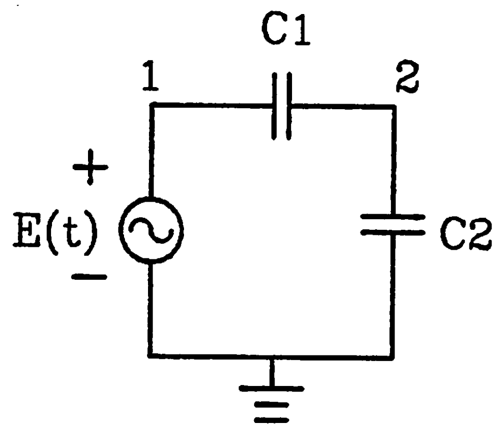


Fig. 2.18 C-C example

$$\Delta V_2 = \Delta V_1 \frac{C_1}{C_1 + C_2} \tag{2.27}$$

A more detailed description of ELogic-3 algorithm will be presented in Chapter 3.

2.13. Area of Future Work

A lot of work has been performed to fully develop the ELogic technique and resolve many of critical issues. In this section, some suggestions for future work are described.

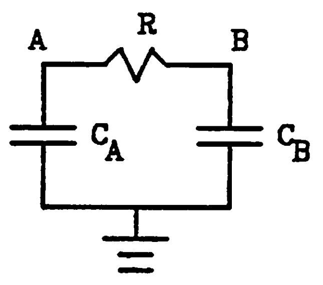
Since only one of the discrete voltage states can be defined as the voltage at a node in the steady state, the node-voltage may alternate between two adjacent states without being triggered by any other nodes. This is called ***self-oscillation*** and it may occur when the exact solution of a node-voltage, V_{EXACT}, is not sufficiently close to the one of the voltage states. Let V_{THRESH} be the threshold voltage between two states V_1 and V_2. If V_{EXACT} lies between V_{THRESH} and V_2 (V_1), it is rounded off to V_2 (V_1). Assume that the present voltage state of node i is V_2 and V_{EXACT} calculated by ELogic is between V_{THRESH} and V_1. Then node i is scheduled to make a transition from V_2 to V_1. When the voltage at node i reaches V_1, V_{EXACT} is computed again. If the new V_{EXACT} is between V_{THRESH} and V_1, the voltage state of node i remains at V_1 unless there is an external trigger by fanin nodes. Under these conditions, the self-oscillation does not occur. However, sometimes the new V_{EXACT} may be between V_2 and V_{THRESH}. This may happen because one of the input voltage states used to evaluate MOS transistors changes from V_2 to V_1. In this case, node i would be scheduled to make a transition back to V_2. As a result, node i alternates between V_1 and V_2. One way to solve this problem is to introduce hysteresis by defining two different threshold voltages, one for rising up (TH_{UP}) and the other for falling down (TH_{DOWN}) of the node voltage, where

$$TH_{UP} = V_{THRESH} + |\delta| , \qquad TH_{DOWN} = V_{THRESH} - |\delta| \tag{2.28}$$

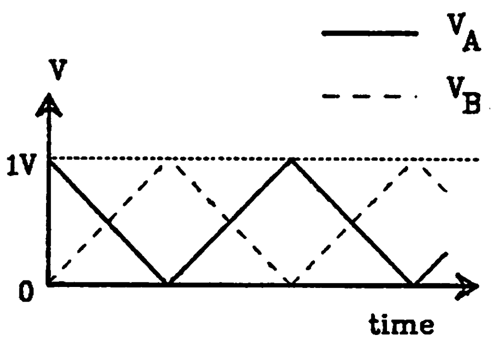
The $|\delta|$ in the above equation must be small enough so that the normal transitions can not be affected. Another solution is to implement a cycle detector, which prevents the node i

from making a transition when the direction of the transition reverses before its fanin node forces it to do so. The latter method is employed in ELogic, although both methods work quite well in practice.

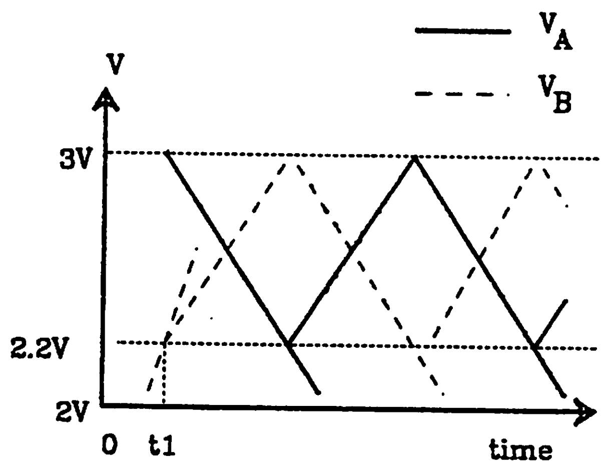
A similar output oscillation may occur due to the interaction of a node and its fanin nodes. This is called ***interactive-oscillation.*** Preventing such interactive-oscillations is very important, since it costs unnecessary CPU-time and it may affect the accuracy of the waveform. The interactive-oscillation can be illustrated using the circuit in Figure 2.19(a). Under certain conditions, node A and B may repeatedly exchange their voltage states with each other as shown in Fig. 2.19(b). Let V_A = 1V, V_B = 0V, $C_A = C_B$ and the voltage step be 1V. At t = 0, both nodes are processed. Node A is scheduled to make a transition to 0V, while node B is scheduled to make a transition to 1V. Since the time constants of two nodes are same, they are scheduled at the same future time point(say, t_1). At t_1, they simply exchange their voltage states and everything else remains same. The same sequence is then repeated over and over. The resulting output waveform is shown in Figure 2.19(b). Since ELogic regards the fanin nodes as the voltage sources, the dual threshold scheme can not solve the problem. For example, at t = 0, node A believes that V_{EXACT} is 0V which is the voltage at B at that time. Similarly, node B believes that V_{EXACT} is 1V which is the voltage at A at that time. A cycle detector is not easy to implement successfully either, since the nodes are scheduled by the fanin node as well as by itself. The interactive-oscillation can occur between one state and any other voltage rather than between two voltage states only. Let $V_A(t_1)$ = 3V, $V_B(t_1)$ = 2.2V, $C_A = C_B$ and the voltage step be 1V. Even though ELogic does not allow 2.2V as a voltage state and does not use it for self-scheduling, it may be used for fanout-adjustments. Assume that node A reaches 3V at t_1, while node B is rising up from 2V to 3V. Then, node B possesses 2.2V which is the node-voltage at B at t_1, and the pending event for B is canceled. After that, node A is processed and scheduled at 2V (self-scheduling) and node B is processed and scheduled at 3V (fanout-adjustment). Since the time constants of two nodes are the same, node A reaches 2.2V when node B reaches 3V. The



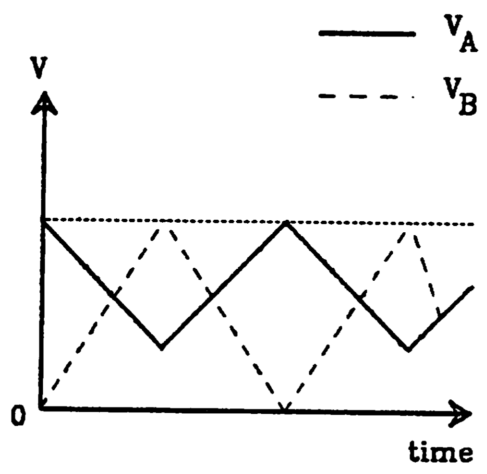
(a)



(b)



(c)



(d)

Fig. 2.19 (a) Example of *interactive–oscillation* (b) Example 1
(c) Example 2 (d) Example 3

procedures are then repeated and their waveforms are shown in Figure 2.19(c). When the value of C_1 is different from that of C_2, the time constants of two nodes are not same. But, if the values of the capacitors are comparable, the interactive-oscillation still may occur. The corresponding waveform is shown in Figure 2.19(d), where the magnitude of the slope of two waveforms are different. Similarly, if each node of pass-transistor chain are electrically similar in nature, the interactive-oscillation may occur. The interactive-oscillation finally dies out, but possibly after a long sequence of the node processes. This costs unnecessary CPU-time and sometimes degrades the waveform accuracy. Therefore, proper handling of interactive-oscillation is one of the most important remaining issues in ELogic.

Another area of future work is to incorporate submatrix methods in the ELogic algorithms, as used in other relaxation-based methods like SPLICE [39, 40] and RELAX [6, 40]. The use of subcircuits improves the convergence speed of the relaxation-based methods in the presence of strong coupling. The subcircuits can be determined either manually or, better, automatically using partitioning heuristics such as [12, 41]. ELogic may dynamically derive the I-G tables for tightly-coupled subcircuits by partitioning and performing subcircuit analysis using direct methods. That is, rather than using static I-G tables, as for common ELogic components, for complex and thus less-often-used subcircuits, the I-G state values would be computed dynamically. Although the submatrix method may be expensive, it solves most problems which arise from the presence of strong coupling, specifically the incorrect decisions of ELogic-1 on transitions and unwanted oscillations due to local coupling. Therefore, the development of a proper submatrix method for the ELogic technique, which can maintain a reasonably low cost, is another important area of research.

2.14. Summary of ELogic Algorithms

So far, the details of three ELogic algorithms have been described. Their basic differences are summarized in Table 2.4.

Algorithm	MOS Model	Floating Cap.	Cap. Model
ELogic-1	Small-Signal	Not Allowed	Const. Vtg. Src. and FE
ELogic-2	Small-Signal Line-Thru-Origin	Not Allowed	Const. Vtg. Src. and TZ
ELogic-3	Small-Signal Line-Thru-Origin	Allowed	Const. Vtg. Src., and TZ

Table 2.4 Summary of ELogic Algorithms

One may wonder which algorithm is the best. The answer depends on the application for which it is to be used. Factors such as accuracy, simulation speed and the limitation of the algorithms must be considered before deciding which algorithm is appropriate for a particular application. The algorithms shown in Table 2.4 are only three examples of the many possible ELogic algorithms. For example, one may use a modified version of ELogic-1 as follows : As indicated earlier, ELogic-1 is the simplest and most efficient algorithm in terms of CPU-time. However, if a node is strongly coupled to another, ELogic may decide that the node does not make a transition for a given voltage step, even though the node can actually make a transition in the circuit. Therefore, if ELogic-1 determines that a node does not make a transition, the voltage step may be reduced temporarily for that particular node. If ELogic-1 consistently claims that the node does not transfer until the voltage step reaches the lower limit, it can be regarded that the node does not transfer in the circuit either. The lower limit of voltage step may be obtained from practical circuits by experiments and it can be stored in the program. By doing so, ELogic-1 can make the correct decision on the transition of the nodes without using the expensive Newton-Raphson iteration.

Another example is a mixed application of two algorithms. One may use ELogic-1 until it determines that a node does not make a transition. Then ELogic-2 may be used to obtain a correct decision on the transition of the node. This algorithm is used in the application to timing verification (Chapter 4).

CHAPTER 3

APPLICATION TO SIMULATION

3.1. Introduction

In this chapter, the ELogic-3 algorithm is described in detail. ELogic-3 is the most expensive of the three algorithms. However, it is also the most general and floating capacitors are allowed in the circuit description. The output waveforms of ELogic are compared with those of electrical simulation, using NMOS and CMOS, and digital and analog test circuits. The CPU-time is also compared for large examples.

3.2. ELogic-3 Algorithm

The stand-alone ELogic-3 program builds two kinds of signal flow graphs, using a linked data structure, as the circuit is read. The first one is a node-signal flow graph which was shown earlier in Figure 2.11. It spreads from node i to its fanout nodes in the structure. This flow graph is used for selective trace technique. The fanin node and the fanout node were defined for the MOS transistor in Figure 2.10. In the case of the resistor and the capacitor, both terminals are fanin as well as fanout nodes of the other. The other signal flow graph which ELogic-3 builds spreads from node j to its fanin nodes which have a direct path to node j. This flow graph is used to obtain the Norton equivalent associated with the node being processed. Not all fanin nodes of node j have a direct path to node j. For example, the gate of the MOS transistor is a fanin node of the source, but there is no direct path between two nodes. Therefore, the gate of the MOS transistor is not linked to the source in the flow graph, unless there is another element which provides a direct path between two nodes. The following is the pseudo-code flow of the general ELogic-3 algorithm for simulation.

```
ELogic_3()
{
/* initialization of hashtables for models and elements */
  hashtable();
/* read the circuit and construct data structure */
  setup();
/* initialize the time queue */
  init();
/* process for entire simulation period */
  proc();
}


setup ()
{
  while(there is input line left) {
    read();
    if(ELEMENT) {
      store in the hashtable(ELEMENT);
      construct two kinds of signal flow graphs();
    }
    else if(MODEL)
      store in the hashtable(MODEL);
    else if(OPTION) set option variable();
    else if(SUPPLY) store the value at the node.
    else if(INPUT_PULSE) store waveform as a linked structure();
    else /* COMMENT */ do_nothing();
  }
/* setup a voltage-state table */
  MakeVtgTable(vtgstep);
/* setup table look-up MOS models */
  MakeMosTable();
}


/* process all nodes at t=0 */
init()
{
 forall (net j) {
   if(PLOT) plot(j);
   if (node_type == SIGNAL) {
     if (get_time(j) == TRUE)
          schedule(j);
     else continue;
   }
   else if (node_type == INPUT)
     schedule j for entire simulation period();
   else do_nothing();
 }
}


proc()
```

```
{
  while(queue is not empty) {
    get node j from the queue;
/* check C-C circuit. refer to Fig. 2.18 */
    if(checkC_C(j) == TRUE) {
      unget(j);
        continue;
    }
    if(PLOT) plot();
    if(node_type == SIGNAL) {
      /* Try self-schedule */
      if(get_time(j) == TRUE) {
        /* checking self-oscillation */
        if(self-oscillation(j) == FALSE) schedule(j);
        }
    }
    /* fanout-adjustment */
    forall(fanout nodes i of j) {
      if (i is active) {
        delete i from the queue;
        if (get_time(i) == TRUE) schedule(i);
      }
      else {
        if (get_time(i) == TRUE) /* activated */
              schedule(i);
      }
    }
  }
}


/* check whether or not there are C-C circuits(Fig. 2.19) */
checkC_C(j)
{
  CFLAG = FALSE;
  if(there is C-C circuits) {
    forall(fanout nodes i of j) {
      calculate the voltage change;
        if(i crosses the voltage state) {
          schedule(i);
          CFLAG = TRUE;
        }
    }
  }
  return(CFLAG);
}


schedule(j)
{
  insert j in the proper position of time queue;
}
```

```
get_time($j$)
{
  if(there is no floating capacitor connected to $j$) {
    use ELogic-2 to calculate transition time;
    return();
  }

  obtain the direction of voltage change at $j$;
  model_capacitors(); /* calculate the transition time */

  if (# of fanin nodes and floating cap., modeled by TZ === 0)
    use ELogic-1 algorithm;

  else if(# of fanin nodes and floating cap., modeled by TZ $\leq$ 2){
    use the root-formula to solve quadratic(cubic) equation;
    if (solution exists) return(TRUE);
    else return(FALSE);
  }

  else { /* use the discretized Newton-Raphson */

  NEWTON_RAPHSON :

  /* get Norton pairs for $T_1$, $T_1 + \Delta T$ */
    Norton($j$,&$I_1$,&$G_1$,&$I_2$,&$G_2$);

  /* discretized N-R */
    disc_NR($I_1$,$G_1$,$I_2$,$G_2$);
    if(CONVERGENT) goto NEWTON-RAPHSON ;
    else if(CONVERGED) return(TRUE);
    else return(FALSE);
  }
}
```

3.3. Analog Circuit Examples

In general, there is relatively little latency in analog circuits. Since latency exploitation is one of the major factors which accounts for the fast simulation speed of ELogic, it has less of a speed advantage over classical simulators for analog circuits than it does for large digital circuits. Most analog circuits usually contain large feedback paths and high gain. Therefore the voltage step must be smaller for analog circuits than for digital circuits, to maintain the same waveform accuracy.

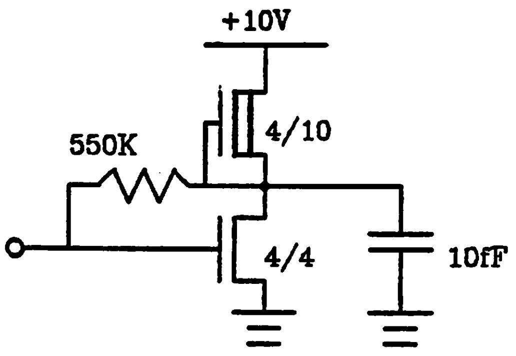


Fig. 3.1(a) Single-Stage NMOS Amplifier with Voltage-Shunt Feedback

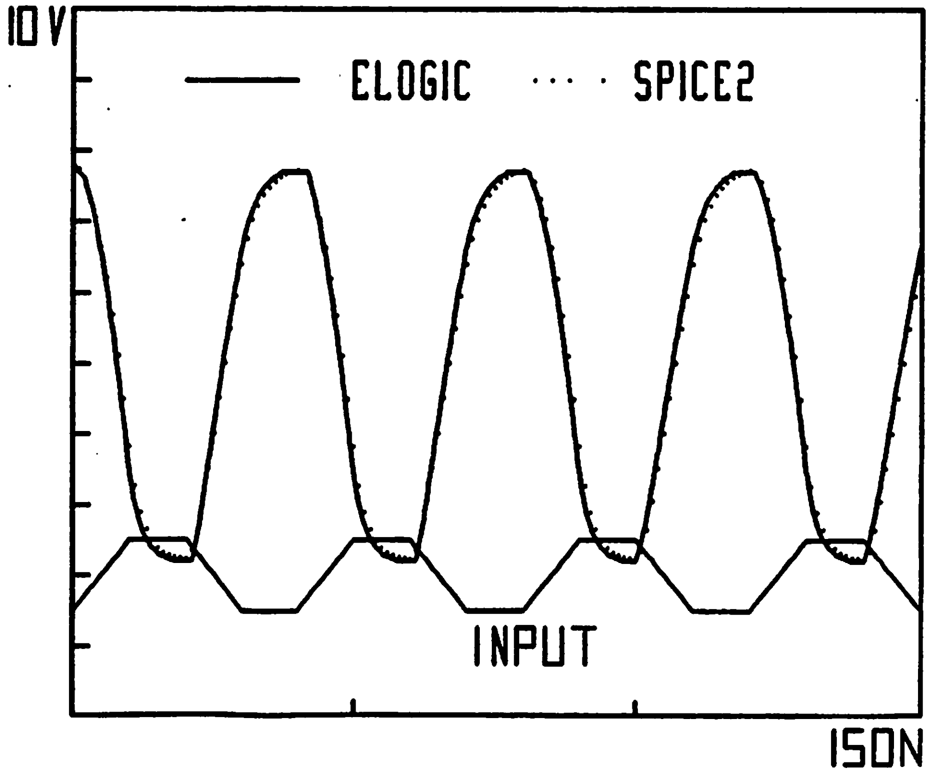


Fig. 3.1(b) Output Waveform of NMOS Single-Stage Amplifier

Four analog circuits, including a single-stage NMOS amplifier, an NMOS ring oscillator, an RC network, and a large industrial NMOS sense amplifier, were chosen to illustrate the capability of the ELogic technique when simulating analog circuits.

3.3.1. Single-Stage NMOS Amplifier

The Single-stage NMOS amplifier with voltage-shunt feedback is shown in Figure 3.1(a). The circuit was simulated using ELogic-3 and small-signal models for MOS transistors with a 0.1V voltage step. The output waveform of ELogic-3 is compared with that of SPICE2 in Figure 3.1(b). The same initial conditions were used by both programs. As the figure indicates, the results are in good agreement.

3.3.2. NMOS Ring Oscillator

The ring oscillator which consists of three NMOS inverters is illustrated in Figure 3.2(a). This circuit has a strong feedback loop. Instead of using external trigger, different initial conditions were assumed at each node to obtain oscillations, as follows.

$$V_1(0) = 0, \quad V_2(0) = 5, \quad V_3(0) = 2.5$$

The output waveform of the last inverter from ELogic-3 is compared to SPICE2 in Figure 3.2(b), for the simulation period of 200 N Sec. The waveform of ELogic-3 was produced using the small-signal model for MOS transistors and a 0.1V voltage step. The relative error toler-

	SPICE2	ELOGIC		
		0.05V	0.1V	0.2V
Time for 10 cycles (NSec)	157.03	156.721	156.136	155.147
Error	-	-0.2%	-0.57%	-1.12%

SPICE2 used 1e-5 as a value of RELTOL

Table 3.1 Timing Comparison of ELogic-3 and SPICE2 for Ring Oscillator (10 Cycles)

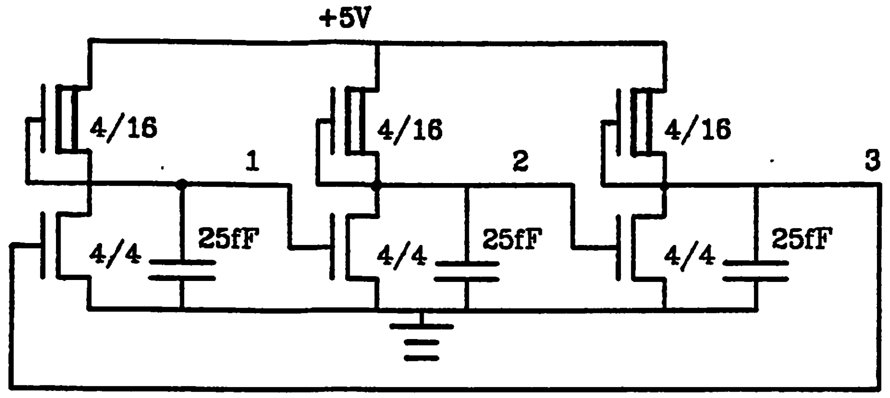


Fig. 3.2(a) NMOS Ring Oscillator

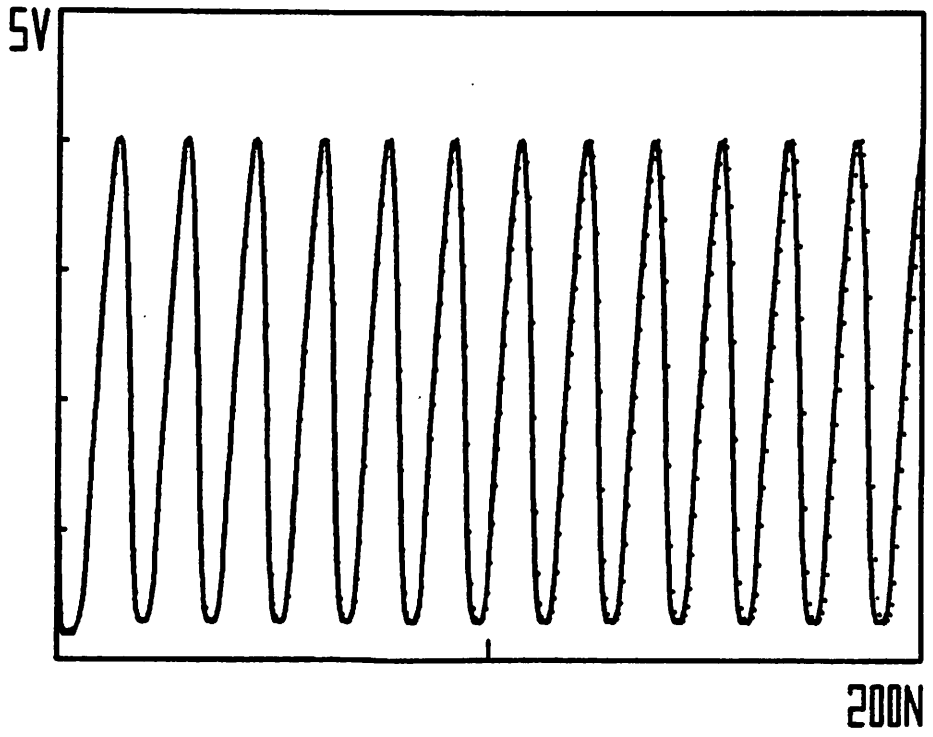


Fig. 3.2(b) The Output Waveform Comparison of Ring Oscillator
ELogic:Solid line SPICE2:Dotted line
ELogic exhibits 0.57% of timing error using 0.1V step

ance (RELTOL) of SPICE2 was set to 1E-5. The two programs produced almost identical output waveforms, in the presence of the strong feedback. The timing error in the output waveform of ELogic decreases further, if smaller voltage is used. To show the dependency of the timing error of ELogic on the voltage step used, the time period of 10 cycles are compared to that of SPICE2 in Table 3.1, for the voltage steps of 0.2V, 0.1V and 0.05V. As the voltage step gets smaller, so does the timing error.

3.3.3. Floating Capacitor Example

An example which contains floating capacitors is illustrated in Figure 3.3. The initial voltages of nodes 2 and 3 are 0V, i.e.,

$$V_2(0) = V_3(0) = 0$$

The waveforms at nodes 2 and 3 using ELogic-3 and SPICE2 are illustrated in Figure 3.4, where ELogic-3 used 0.25V step. The solid lines are ELogic-3 output waveforms and the dotted lines are SPICE2 output waveforms. For the node 2, the waveform of SPICE2 and ELogic-3 are in good agreement. This suggests that ELogic-3 handles the floating capacitor successfully. Remember that one of the major drawbacks of standard timing simulators is their inability to handle floating capacitors. In the case of node 3, the waveform of ELogic falls behind that of SPICE2 in the beginning. This is because in ELogic, node 3 starts to make a transition only after node 2 makes its transition. This is not due to the floating capacitor but rather it comes from the fact that ELogic uses the information at present time point to calculate the next time point. ELogic also produces a similar time delay error for an RC chain and pass-transistor chain circuits. The error certainly depends on the voltage step used and the depth of nodes from the input source. The smaller the voltage step is, the smaller will be the error. And, the nearer to inputs the node is, the smaller will be the error. Even though this is a major drawback of the ELogic technique, the voltage error dies out at steady state except for the usual discretization voltage error. Figure 3.4(b) indicates that there is only the discretization voltage error at node 3 after 10NSec. Most large digital circuits

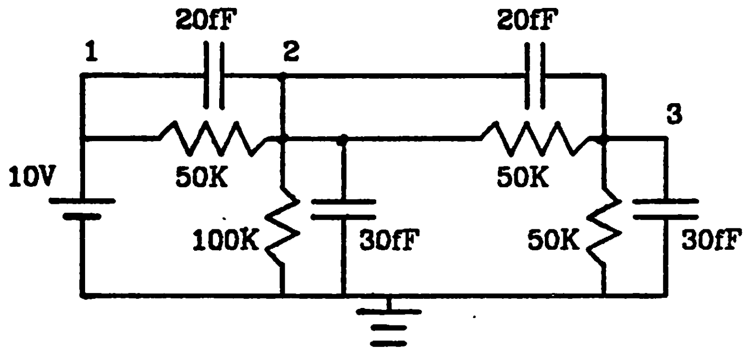


Fig. 3.3 RC Network with Floating Capacitor

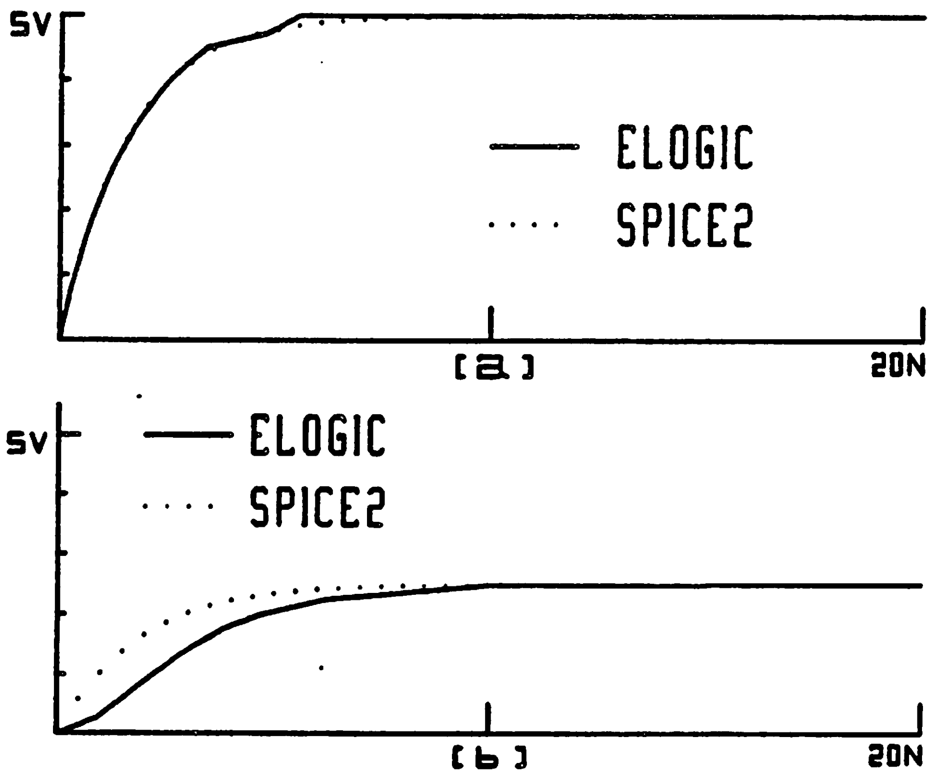


Fig. 3.4 Waveform of ELogic-3 and SPICE2 (a)at node 2 (b)at node 3
ELogic-3 used 0.25V step.

employ a lot of regenerative circuits such as latches and flip-flops. Therefore the delay error does not accumulate while signal propagates from the input node to the output node, in practice.

3.3.4. Industrial Sense Amplifier

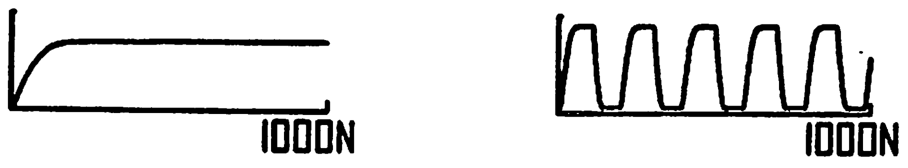
An industrial sense amplifier which contains 723 NMOS transistors and 401 nodes was simulated for 1000 N sec on VAX[1] 11/780-5 running UNIX[2], by four programs including SPICE2, SPLICE2 [2], ELogic-1, and ELogic-2 to compare output waveforms and CPU-time. Figure 3.5 illustrates the output waveforms of the four programs for two active nodes (node 14 and 41). ELogic-1 and ELogic-2 used 0.2 V step and the small-signal model. Both ELogic-1 and ELogic-2 produced acceptable output waveforms for all nodes. However, it was observed that ELogic-2 presented a slightly better output waveform at node 14 than ELogic-1. The CPU-time is compared in Table 3.2. The stand-alone ELogic-1 and ELogic-2 programs are 188.3 times and 75.8 times faster than SPICE2, using 0.2V step, respectively.

Program	CPU-time(Sec)	Speed (Normalized to SPICE2)
SPICE2	9542.6	1
SPLICE2	1067	9
ELogic-1	50.68	188.3
ELogic-2	124.61	75.8

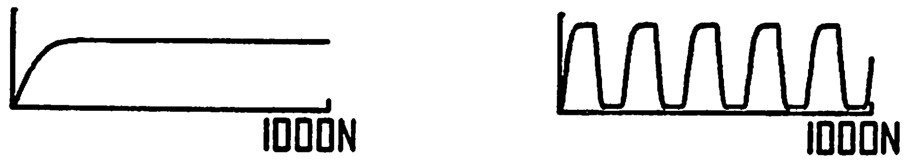
Table 3.2 CPU-time Comparison for NMOS Sense Amplifier

[1] VAX is a trademark of Digital Equipment Corporation

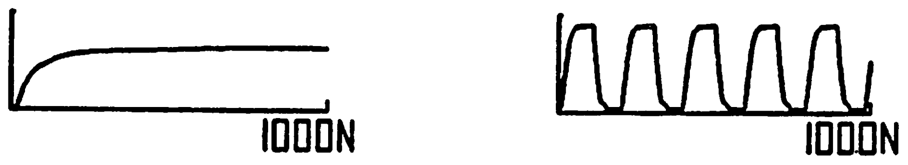
[2] UNIX is a trademark of Bell Laboratories.



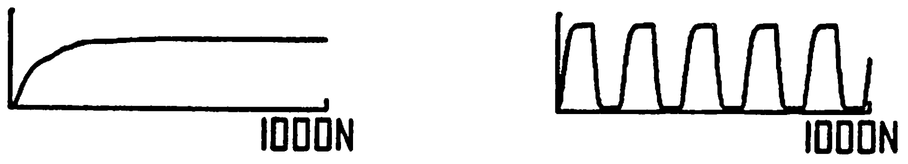
[a]



[b]



[C]



[d]

Fig. 3.5 Waveforms of NMOS Sense Amp. (Node 14 (Left),Node 41(Right))
(a) SPICE2 (b) SPLICE2 (c) ELogic-1(0.2V step) (d) ELogic-2(0.2V step)

3.4. Digital Circuit Examples

3.4.1. NMOS Inverter Chain

The 3-stage NMOS inverter chain is shown in Figure 3.6. The inverter chain was simulated by ELogic-3 using a small-signal model for two voltage steps : 5V and 0.25V. Figure 3.7 illustrates ELogic output waveforms at each node, where ELogic used 5V voltage step. Since only two voltage states, 0V and 5V, are allowed, the output waveforms look similar to the output waveforms of a logic simulator. Note that the output of each inverter starts to change only when its input waveform achieves either 0V or 5V. Figure 3.8 illustrates how the precision of the ELogic model affects the accuracy of ELogic waveform, by comparing its waveform at node 4 to that of SPICE2, for two voltage steps of 0.25V and 5V. In the figure, 0.25V step presents a much closer output waveform to that of SPICE2 than 5V step. However, it certainly costs more than the 5V step case. Therefore, the voltage step must be

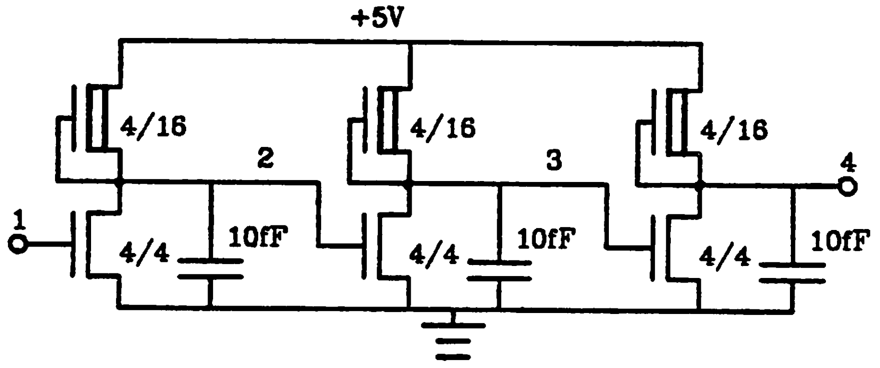


Fig. 3.6 NMOS Inverter Chain

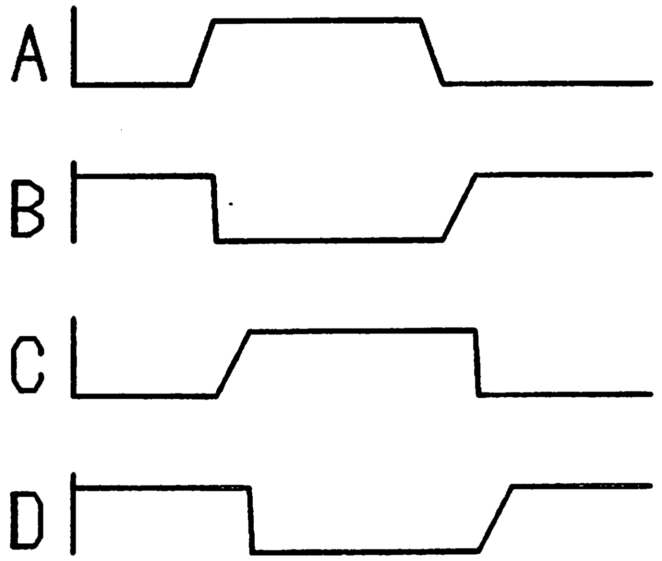


Fig. 3.7 ELogic output waveforms using 5V step
(A) input (B) node 2 (C) node 3 (D) node 4

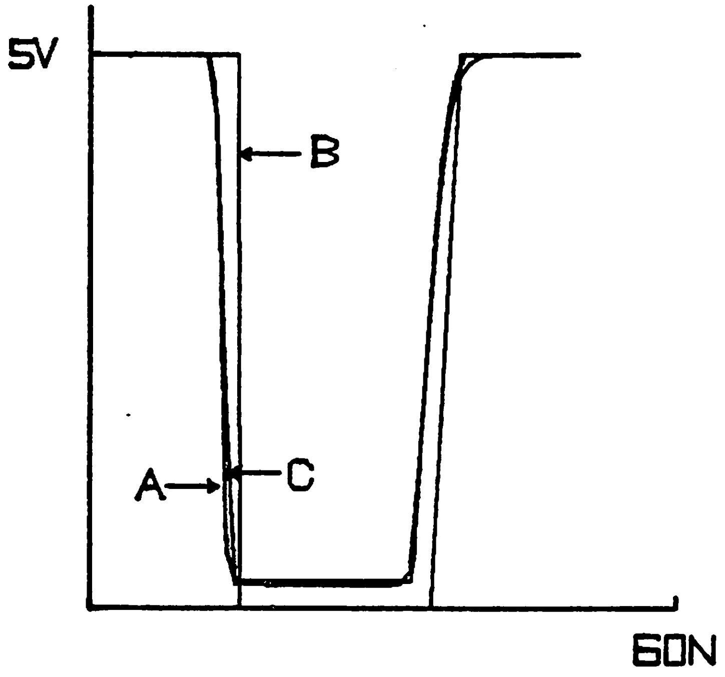


Fig. 3.8 Waveform Comparison between ELogic and SPICE2
(A) ELogic(0.25V step) (B) ELogic(5V step) (C) SPICE2

chosen appropriately for the purpose of the application, in terms of accuracy and speed.

3.4.2. CMOS Static RAM Cell

As a CMOS circuit example, a commonly used CMOS static RAM cell is illustrated in Figure 3.9. The cell employs a pair of crossed-coupled CMOS inverters as the storage flip-flop. The row select line R is held low except when the cell is to be accessed for writing or reading. The C and $\overline{C}$ lines are biased at approximately 3V, initially. Writing and reading are performed through the column lines, C and $\overline{C}$, and transistors $M3$ and $M4$. The data is read when M3 and M4 are turned on by keeping the row select line R high. For fast read-response, the small voltage change at C and $\overline{C}$ is amplified by sense amplifiers(not shown). Writing is accomplished by forcing either C or $\overline{C}$ low through buffer, depending on the data

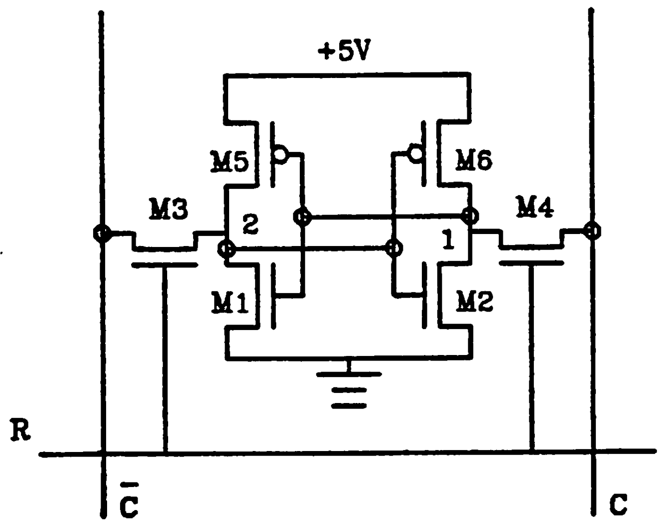


Fig. 3.9 CMOS Static RAM Cell

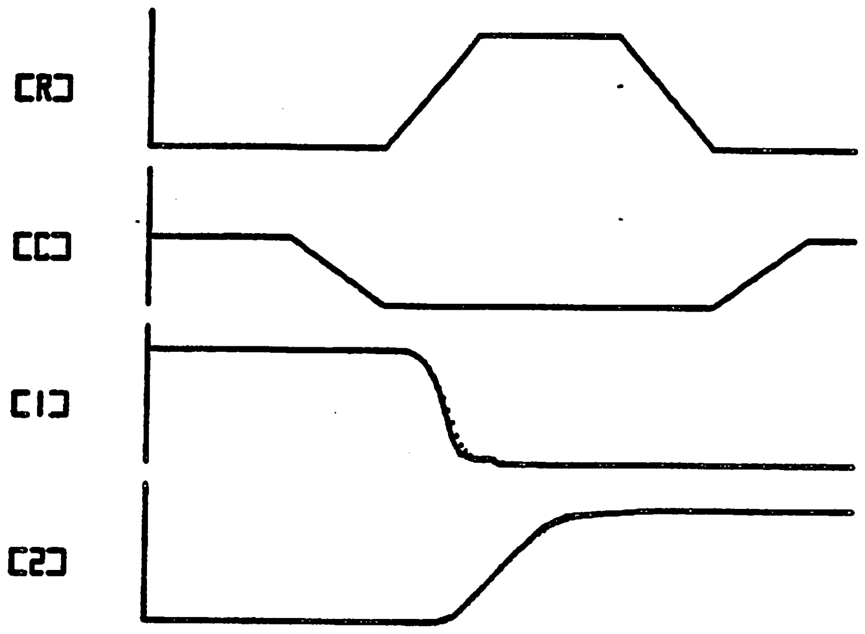


Fig. 3.10 Waveforms when Writing CMOS Static RAM Cell (Node R,C,1, and 2)
ELogic(0.25V step):Solid Line SPICE2:Dotted Line

to be stored. In Figure 3.10, the writing waveforms are compared using SPICE2 and ELogic-3, when C is forced to ground from 3V. In the test, ELogic-3 used a voltage step of 0.25V and modeled $M3$ and $M4$ by the line-thru-origin model and other transistors by the small-signal model. In Figure 3.10, the output waveforms of ELogic-3 is very close to that of SPICE2,and the waveforms are almost indistinguishable.

3.4.3. ALU of CMOS SOAR (Smalltalk On A RISC)

As an example of a large digital circuit, the ALU of the CMOS SOAR (Smalltalk On A RISC) chip [46] was simulated on VAX 7800 running UNIX using SPLICE2, ELogic-1, and ELogic-2 for 300 N Sec, to compare output waveforms and CPU-time. For the analysis parameters, ELogic used the small-signal model and a 0.2 V step. The circuit is designed

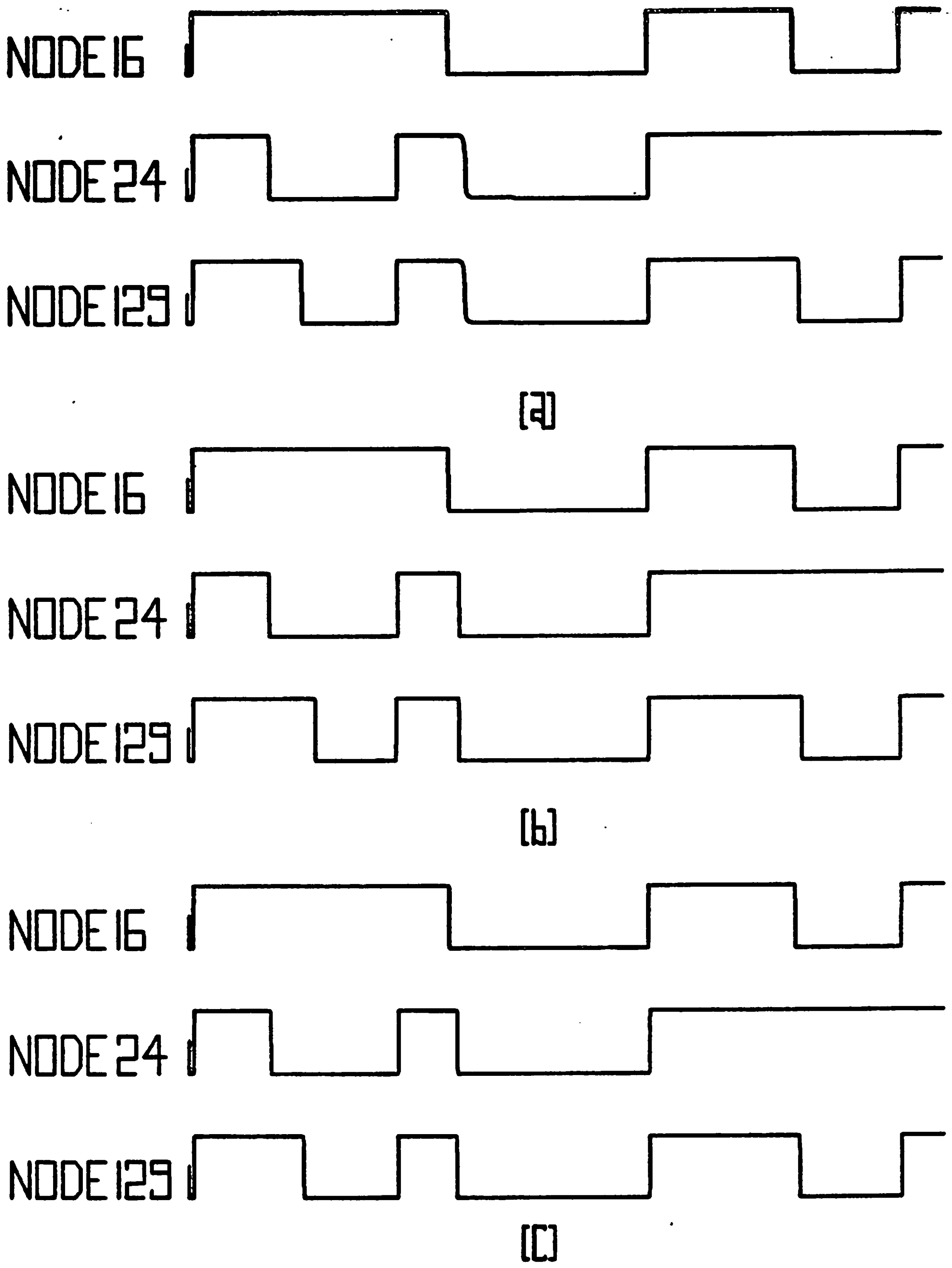


Fig. 3.11 Output Waveforms of SOAR ALU
(a) SPLICE2 (b) ELogic-1 (c) ELogic-2

using Domino logic and contains 1692 transistors and 1050 nodes. Waveforms at some output nodes are illustrated in Figure 3.11. It is observed that both stand-alone ELogic programs produced acceptable waveforms. The CPU-time are given in Table 3.3.

Program	CPU-time (Sec)	Speed (Normalized to SPLICE2)
SPLICE2	1131	1
ELogic-1	346.6	3.26
ELogic-2	512	2.2

Table 3.3 CPU-time Comparison for CMOS SOAR ALU

For this particular example, ELogic-1 and ELogic-2 do not present much speed improvement over SPLICE2 as they did for NMOS sense amplifier (Refer to Table 3.2). It is because the scheduling scheme implemented in the stand-alone programs always searches from the first point of the queue and not suitable for the circuit with a large number of nodes. The fanin-oscillation (refer Section 2.13) also increased the cost of the ELogic simulation.

3.5. Conclusion

In this chapter, the flow of the ELogic-3 algorithm has been described in detail. The algorithm successfully solves a circuit which contains floating capacitors. Experimental results indicate that ELogic can successfully simulate both analog circuits and digital circuits. It suggests that ELogic can be used to simulate a circuit which contains both digital and analog parts by using different voltage steps for each part depending on the circuit nature. For analog circuits, small voltage steps are necessary to maintain accuracy in the waveforms, due to the presence of strong feedback and high gain. For digital circuits, much larger voltage steps may be used. Since the ELogic techniques were implemented to illustrate their capability of simulation in the stand-alone programs, it is possible to improve the speed of the ELogic programs by further optimization.

CHAPTER 4

APPLICATION TO TIMING VERIFICATION

4.1. Introduction

Digital circuits are designed in the form of either asynchronous systems or synchronous systems. For asynchronous systems, great care must be exercised to avoid potential timing errors due to any difference in state sequencing under arbitrary differential delays of signals through the paths in the circuit, during the design process. The synchronous system fits into the clocking scheme and implementation functions correctly only if the clock period is sufficiently longer compared to the delays in the circuit. Therefore, the speed of the circuit is determined by the slowest of all possible signal paths, which is called *critical-path*. On the other hand, unnecessarily fast sections of the circuit consume extra power. Designers attempt to verify the timing of the circuit and correct timing errors in the circuit while optimizing speed-power product. This is becoming a more and more difficult task as circuit size and complexity increase and the growing competition for high performance circuits makes circuit ever more critical. Simulation is a very inefficient technique for this purpose. Furthermore, pathological conditions may not be detected by simulation, unless particular inputs are fed. Therefore, *timing verification* has played a very important role in designing the digital integrated circuit and in optimizing its performance [43, 44, 48, 49].

There are two approaches in timing verification [42], called *path enumeration* and *critical-path analysis*. Path enumeration [47] checks all possible paths in the circuit for timing problems, while in the critical-path analysis the search is pruned and only the slowest paths are detected. Path enumeration is conceptually simple, but it may suffer from rather long CPU-times due to the potentially large number of possible paths. Even though the major difficulty of the critical-path analysis is its implementation complexity, it has been successfully implemented and applied to MOS switch-level circuits in programs such as Crystal [43] and TV [44]. In this report, ELogic is used in conjuction with the critical-path analysis approach.

Unlike simulation, the critical-path analysis is *value-independent*. This means that, for example, if a changing signal comes into a NAND gate, its effect is always propagated to the output node, regardless of the signal states at the other input terminals. In simulation, the changing signal propagates to the output of the NAND gate, if and only if the signal states at the other input terminals support the propagation. Both the strength and weakness of critical-path analysis comes from the value-independence. It enables the critical-path analysis to avoid checking for all possible input vectors, which grows exponentially with the number of input nodes. As a result, the critical-path analysis can save much CPU-time, compared to simulation. However, since the critical-path analysis ignores specific signal values, it may report critical paths which can never occur under real operating conditions, which are called *false paths*. The false paths tend to hide the real critical paths in the circuits under test. In this case, a mechanism called case analysis, which fixes the node value, may be used. However, it must be used with caution so that critical path can not be overlooked. In general, case analysis must be run several times to make sure, with different values each run.

While some timing verification programs [48] model the circuit at gate-level, switch-level timing verifiers [43, 44] represent each nonlinear transistor by a linear resistor in series with a switch. Then they calculate the delay time using one of a number of different RC approaches. The RC approaches use RC time constants for computing delays, rather than performing simulation. These RC approaches are very efficient in terms of CPU-time, but they have following weakness. They assume that there is only one direct path from a reference node (power supply or ground) to the signal-nodes of the circuit. For example, in the case of NMOS inverter, if the driver is on, only the driver is considered for delay time estimates, and the load is ignored even though it is on too. While some of the RC approaches incorporate information about input waveform shape and load condition in order to obtain more accurate delay estimates, they use the ratio approach which was first suggested by Pilling and Skalnik [45]. In the ratio approach, factors such as the input rise time, the output load, and transistor size are combined into single ratio, called *rise-time ratio*. This value is then used to determine the effective resistance of a transistor. The ratio approach improves the accuracy of the delay estimates significantly with small amount of work for most circuits, it may still result in large error. The RC approaches do not provide an accuracy-speed trade-off either.

In this chapter, the implementation of ELogic as a delay modeler in the switch-level timing verification program *Crystal* [19, 20, 43] is described. Crystal provides the designer with information about the time required for the output nodes to settle and the worst-case paths leading from the input to those nodes. The experimental results are also included.

4.2. Path Analysis and Delay Modeler

In general, timing verification programs are partitioned into two sections called the path-analysis section and the delay modeler, as illustrated in Figure 4.1. Most timing verifiers represent a circuit by a node-signal flow graph as shown previously in Figure 2.11. The path-analysis section extracts part of the circuit systematically using the signal flow graph and transfers it to a delay modeler so that the delay modeler can compute its delay. Using the delay information obtained from the delay modeler, the timing verifier locates the critical paths of the circuit.

Crystal decouples the circuit into chains of transistors called *stages*. Since a single transistor can participate in many different stages during a single timing verification run, Crystal extracts all the stages triggered by a change at a node dynamically during timing verification, and passes each stage to a delay modeler. Delay modeler computes how long it will take for the output of the stage to change value, by using an RC approach. Once the stage has been processed, it is discarded. Crystal uses a depth-first search to trace out stages

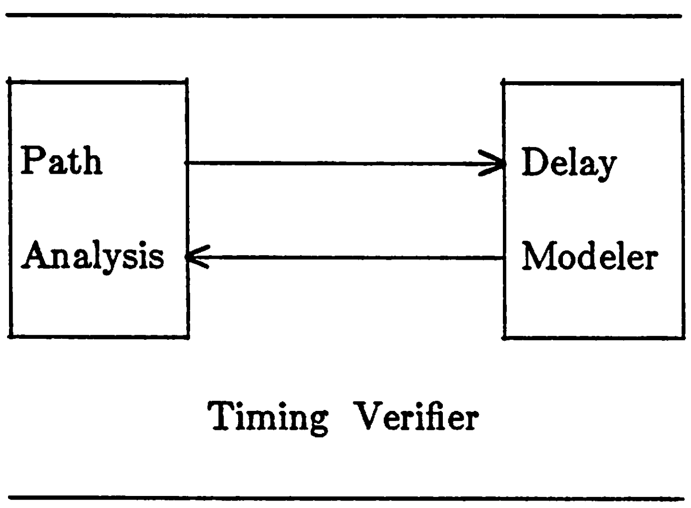


Fig. 4.1 Partition of Program Structure of Timing Verifier

and locate the critical paths.[1] Crystal stores in each node the slowest delay time to that node that has been found so far. When a node appears as the output of a stage, the new delay time from the stage is compared to the time stored in the node. Then the slower time is stored in the node. Since delay modeler is separate from the path analysis section in the timing verification program, ELogic can be used to calculate the delay time of the stages. While the RC approaches have been used successfully in designing large digital MOS circuits, ELogic can provide more accurate delay estimates. In addition, ELogic provides a continuous accuracy-speed trade-off which is one of the major features of its applications.

[1] Jouppi's TV program [44] uses a breadth-first search.

4.3. Implementation of ELogic Model in Crystal

Crystal is written in "C" programming language. The stand-alone ELogic delay modeler was added in Crystal with emphasis on consistency of the program structure and minimum modification of the existing code rather than performance optimization. Since ELogic is a simulation technique, a *pointer* was added into the data structure for the stage to store the waveform segment of its output node. Note that Crystal assumes each gate of the transistors in a stage, except an input node, can take only one of two states, i.e., *Vdd* or ground. Since the version of Crystal employing ELogic modeler uses the same structure, the simulation is not ideal.

When Crystal calls the delay modeler, it transfers information about the stage to the delay modeler. Since each stage is a chain of transistors from the signal source to a gate or output node, the stage, as is, may not be the correct subcircuit to simulate using the ELogic technique. In this case, a signal source is any strong source of a logic 0 or 1. For example, if *Input* of Figure 4.2 turns on, the following four stages are created for delay evaluation : M2-M3-M4, M2-M3-M5, M1-M3-M4 and M1-M3-M5. It is observed that either M1 or M2 is missing for the stages. Therefore, a function called *StageSetup()* was implemented to complete the stage by tracing out missing parts from input file, if there are any.

The definition of the delay was slightly modified from the conventional definition of the propagation delay. The propagation time is the time difference between 50% points of the input and output pulse waveforms. However, the delay time used by Crystal is based on the typical logic threshold voltage, V_{inv}, at which the input voltage is equal to the output voltage of the inverter. Once the output waveform segment is obtained for a stage, the segment is stored in the output node i. Later, the stored output waveform segment is used as an input waveform segment for the stage whose input node is i. The simulation using ELogic is performed until the output waveform reaches a steady state.

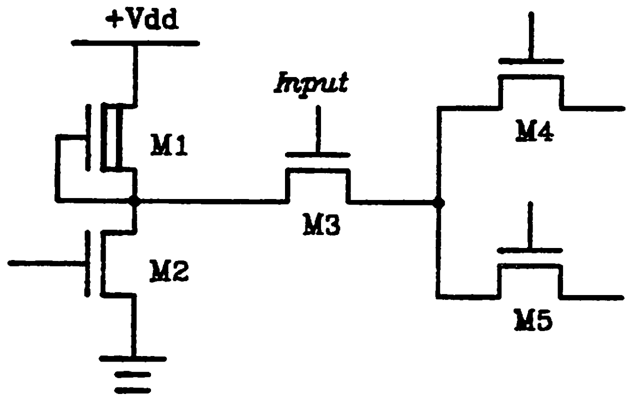


Fig. 4.2 Circuit Fragment

The ELogic modeler was implemented using a mixed algorithm of ELogic-1 and ELogic-2 (Mixed-ELogic). When it evaluates the delay time, ELogic-1 is used first for each transition. If a node fails to make that transition for some reasons, perhaps due to the use of large voltage steps, the algorithm switches from ELogic-1 to ELogic-2 which guarantees that the node will make a transition, if appropriate. The high-level pseudo-code algorithm is as follows.

```
/* evaluate the delay by ELogic technique */
ELogicDelay(stage)
{
/* complete the stage for ELogic */
  StageSetup(stage);

/* process all nodes at beginning */
  init(stage);

/* obtain the waveform at output node */
  while(the queue is not empty OR output node is not in steady state) {
        get next node $j$ form the queue;
        node_proc($j$);
        forall(fanout nodes $i$ of $j$)
            node_proc($i$);
    }
    return(delay);
}

node_proc($j$)
{
    use ELogic-1();
    if ($j$ made a transition)
        return();
    else
        use ELogic-2();
}
```

4.4. Examples and Results

The ELogic model was evaluated using a CMOS microprocessor SOAR (Smalltalk On A RISC) chip[46] and its ALU (Arithmetic Logic Unit).

4.4.1. ALU of CMOS SOAR(Smalltalk On A RISC)

To observe the effect the effect of voltage step change of ELogic and the difference between ELogic-1 and Mixed-ELogic, the ALU part of SOAR was extensively evaluated. The circuit had been used as an example for the application of ELogic to simulation, in Chapter 3. Crystal with each delay model was run on the circuit to extract 15 critical paths.

Figures 4.3 and 4.4 illustrate the comparison between the Mixed-ELogic model of 10 states(0.5V step), the Mixed-ELogic model of 25 states(0.2V step), and the Crystal distributed slope model to SPICE2. In the figures, the X and Y axes represent the delay estimates by SPICE2 and by delay modeler (Mixed-ELogic or distributed slope model), respectively. The straight line illustrates the delay estimates comparison between distributed slope model and SPICE2. The dotted line compares between Mixed-ELogic and SPICE2. Each line of the figures corresponds to one critical path. Each segment of the line corresponds to a stage in the path. The breakpoint represents the delay estimates up to that stage. If delay estimates

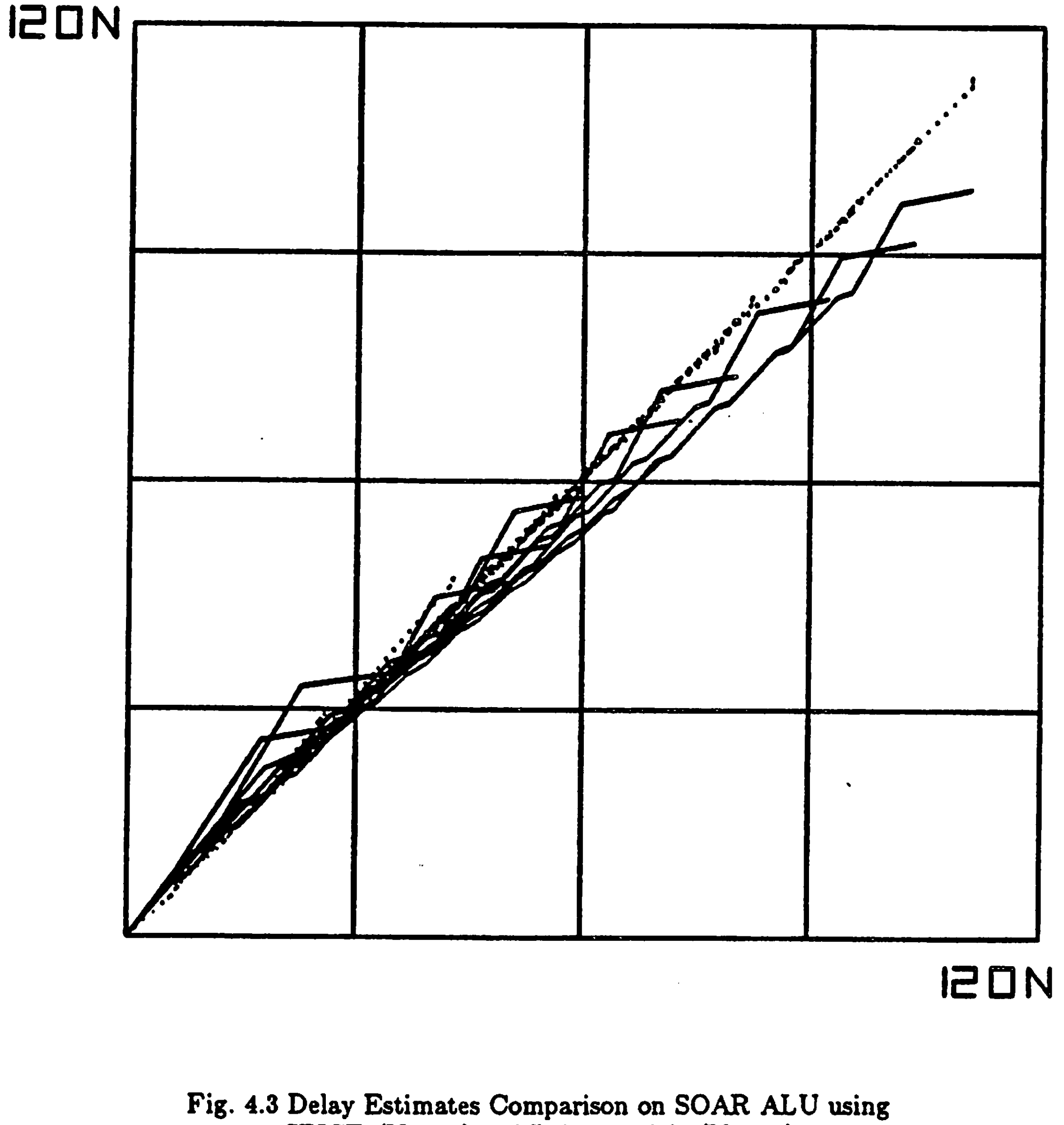


Fig. 4.3 Delay Estimates Comparison on SOAR ALU using SPICE2(X-axis) and Delay modeler(Y-axis)

Solid line : SPICE2 and Distributed Slope Model
Dotted line : SPICE2 and Mixed-ELogic (0.5V step)

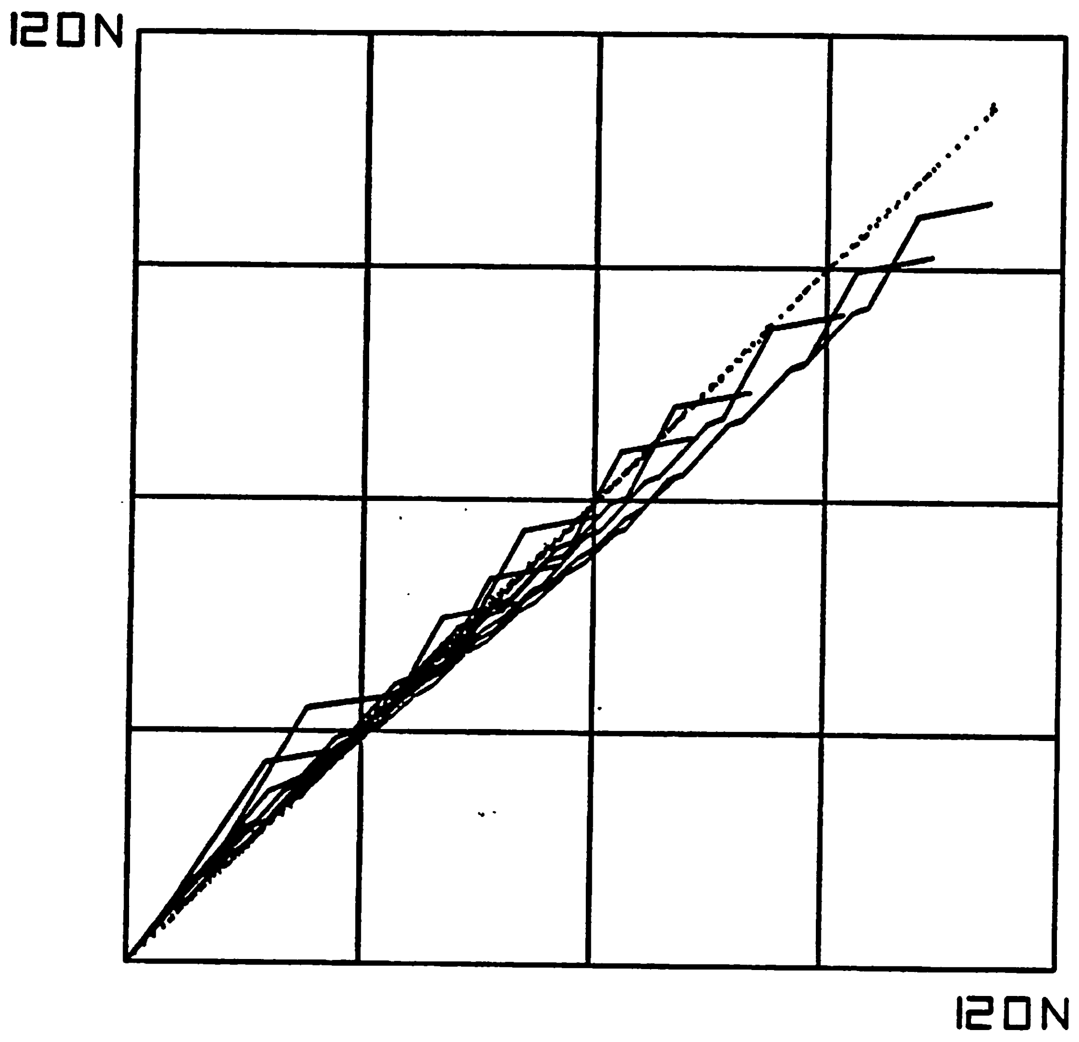


Fig. 4.4 Delay Estimates Comparison on SOAR ALU using SPICE2(X-axis) and Delay modeler(Y-axis)

Solid line : SPICE2 and Distributed Slope Model
Dotted line : SPICE2 and Mixed-ELogic (0.2V step)

of a path by the delay modeler is exactly the same as those by SPICE2, it appears as a 45° line from left bottom to right top corner. The results for the ALU are summarized in Table 4.1. In this table, the "Overall Error" is an error of total delay estimates obtained for all critical paths by delay modeler compared to that of SPICE2. The "Avg. Stage Error" is the average of the absolute error for stages, compared to SPICE2. Similarly, "Avg. Path Error" is the average of the absolute error in estimating total delays in the critical paths up to each stage. The last two columns of the table illustrate the CPU-time required by using each modeler, on VAX 8600. "Delay Eval." is the CPU-time required only to evaluate the delay through paths, and "total CPU-time" includes all other work such as reading the input file and constructing data structures.

From Figures 4.3 and 4.4, and Table 4.1, it is observed that ELogic provides more accurate delay estimates than the distributed slope model. It is also observed that the accuracy and CPU-time depend on the algorithm and the voltage step used. Even though Mixed-ELogic is more expensive than ELogic-1, for same voltage step, it is more accurate too. For example, Mixed-ELogic with 0.2V step used almost same CPU-time as ELogic-1 with 0.1V

Model	Overall Error(%)	Avg. Stage Error(%)	Avg. Path Error(%)	CPU-Time(Sec)	
				Delay Eval.	Total
Distributed Slope Model	5.5	8.5	7.2	0.7	3.5
ELogic-1 (0.1V step)	-1.0	2.3	1.2	225	238
ELogic-1 (0.2V step)	1.2	4.9	3.9	106	110
Mixed-ELogic (0.1V step)	0.78	1.1	0.5	541	544
Mixed-ELogic (0.2V step)	-0.45	1.3	2.0	243	247
Mixed-ELogic (0.5V step)	1.0	1.8	3.5	159	163

Mixed-ELogic is a mixed algorithm of ELogic-1 and ELogic-2

Table 4.1 Delay estimates and CPU-time Comparison on SOAR ALU

step, but it provided more accurate delay estimates.

4.4.2. SOAR Chip

12 critical paths of the SOAR chip were extracted by using Crystal. Delays through these paths were estimated by using Mixed-ELogic, the distributed slope model, and SPICE2. The results are given in Table 4.2 as well as Figure 4.5.

Model	Overall Error(%)	Avg. Stage Error(%)	Avg. Path Error(%)	CPU-Time (Sec)
Distributed Slope Model	-19.3	34.5	46.4	0.028
Mixed-ELogic (0.1V step)	-1.2	19.6	1.6	14.47
Mixed-ELogic (0.2V step)	0.45	20.7	4.2	7.51
SPICE2	-	-	-	170.85

Mixed-ELogic is a mixed algorithm of ELogic-1 and ELogic-2

Table 4.2 Delay estimates and CPU-time Comparison on SOAR

Notice that the path "A" in Figure 4.5, which is an actual critical path by SPICE2, may be hidden by other paths by using distributed slope model. Since ELogic is more accurate, it can reduce the chance of hiding the critical path, as illustrated.

4.5. Conclusion

It has been verified experimentally that ELogic can be used successfully as a delay modeler in timing verification. The experimental results also illustrate that the accuracy of the delay estimates was much improved over other existing techniques.

Crystal uses a depth-first analysis, which has an advantage in handling circuits with cycles in a natural way. However, the modeler may be applied to each node more than once. Since the delay modeler employed by the original Crystal is very fast, it was not a significant issue. If the ELogic modeler had been implemented in breadth-first analysis and the stage

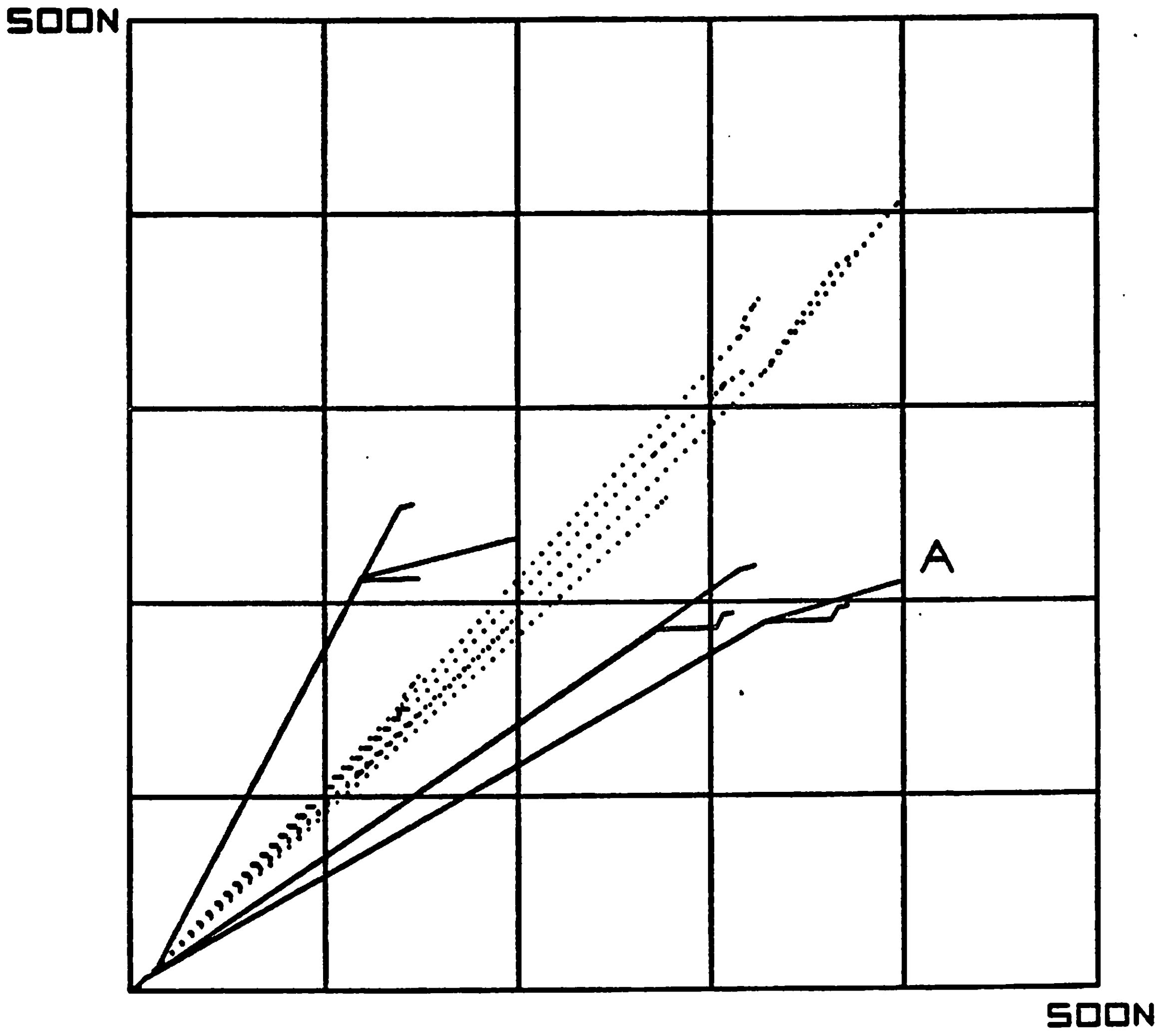


Fig. 4.5 Delay Estimates Comparison on SOAR using SPICE2(X-axis) and Delay modeler(Y-axis)

Solid line : SPICE2 and Distributed Slope Model
Dotted line : SPICE2 and Mixed-ELogic (0.2V step)

information had been extracted more suitably for an ELogic model, the delay modeler may have been applied only once for each node and the total run time could have been reduced significantly. The basic concept of timing analysis as a value-independent approach was not modified and some limitations of the approach remain, such as a possible reporting of false critical path. The study of the new approach which can avoid the false critical path automatically, by taking an advantage of that ELogic is a simulation technique, would be worthwhile.

CHAPTER 5

CONCLUSIONS

Electrical–Logic simulation (*ELogic*) is a new relaxation-based switch-level simulation technique, which solves for the time required for a node to make a certain voltage change rather than solving for a node voltage at a given time point. Since the number and/or values of the voltage states that can be specified may vary, ELogic provides for a continuous accuracy-speed trade-off. Three ELogic algorithms, ELogic-1, ELogic-2, and ELogic-3, have been described in detail so that an appropriate algorithm can be chosen and modified as necessary, depending on the application. Factors such as accuracy, simulation speed and the limitation of the algorithms must be considered.

The comparison of output waveforms of ELogic to existing electrical simulators showed that ELogic can simulate both analog and digital MOS circuits quite accurately. ELogic also provided a speed improvement. In addition to its application to simulation, It was shown that ELogic can be used as an accurate delay modeler of switch-level timing verifier.

REFERENCES

1. A. R. Newton, "Timing,Logic and Mixed-Mode Simulation for Large MOS IC," *NATO Advanced Study Institute on CAD for VLSI Circuits*, University of California, Berkeley, (June - Aug 1980).

2. J. E. Kleckner, "Advanced Mixed-Mode Simulation Techniques," *UCB/ERL M84/48*, University of California, Berkeley, (June 1984). Ph.D. Dissertation

3. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," *UCB/ERL M75/520*, University of California, Berkeley, (May 1975). Ph.D. Dissertation

4. , "Advanced statistical analysis program (ASTAP)," Pub. No. SH20-1118-0, IBM Corp. Data Proc. Div., White Plains, NY ().

5. Resve A. Saleh, "Iterated Timing Analysis and SPLICE1," *UCB/ERL M84/2*, University of California, Berkeley, (Jan 1984).

6. J. White and A. Sangiovanni-Vincentelli, "RELAX2: A New Waveform Relaxation Approach for the Analysis of LSI MOS Circuits," *Proc. 1983 Int. Symp on Circ. and Sys.*, (May 1983).

7. F. Yamamoto and S. Takahashi, "A Vectorized LU Decomposition Algorithm for Large Scale Circuit Simulation," *Digest 1984 Int. Conf. on CAD*, (Nov 1984).

8. A. Vladimirescu and D.O. Pederson, "Circuit Simulation on Vector Processors," *Proc. ICCC 82*, (1982).

9. Beyers, J.W., "A 32-bit VLSI CPU Chip," *ISSCC Digest of Technical Papers*, pp. 104-105 (Feb 1981).

10. J. E. Kleckner, R. A. Saleh , and A. R. Newton, "Electrical Consistency in Schematic Simulation," *Proc. IEEE Int. Conf. on Circ. and Comp.*, pp. 30-34 (October 1982).

11. A. R. Newton and D. O. Pederson, "Analysis Time,Accuracy and Memory Requirement Tradeoffs in SPICE2," *Proc. Asilomar Conference*, (1978).

12. J. White and A. Sangiovanni-Vincentelli, "Partitioning Algorithms and Parallel Implementations of Waveform Relaxation Algorithms for Circuit Simulation," *Proc. 1985 Int. Sym. of Circuits and Systems*, (June 1985).

13. Resve A. Saleh, James E. Kleckner, and A. Richard Newton, "Iterated Timing Analysis in SPLICE1," *Digest 1983 Int. Conf. on CAD, IEEE*, (Sept 1983).

14. R.E. Bryant, "An Algorithm for MOS Logic Simulation," *LAMBDA*, pp. 46-53 (4th Quarter 1980).

15. Young Hwan Kim, J. E. Kleckner, Resve A. Saleh, and A. R. Newton, "Electrical-Logic Simulation," *Digest 1984 Int. Conf. on CAD, IEEE*, (Nov 1984).

16. , *LOGIS: User's Manual Version 4*, ISD Corporation (1980).

17. F. Jenkins, *ILOGS: User's Manual*, Simutec (1982).

18. C.M. Baker and C. Terman, "Tools for Verifying Integrated Circuit Designs ," *LAMBDA*, (4th Quarter 1980).

19. J. K. Ousterhout, *A Switch-Level Timing Verifier for Digital MOS VLSI*, University of California, Berkeley (Feb 20, 1985).

20. J. K. Ousterhout, *Using Crystal for Timing Analysis*, University of California, Berkeley (Sept 12, 1983).

21. DeMan, Arnout, and P. Reynaert, "Mixed-Mode Circuit Simulation Techniques and their Implementation in DIANA," *NATO Advanced Study Institute on CAD for VLSI Circuits*, (July 1980).

22. K.A.Sakallah, "Mixed Simulation of Electronic Integrated Circuits," *DRC-02-07-81*, Carnegie Mellon University, (Nov 1981).

23. H. Schichman and D.A. Hodges, "Modeling and Simulation of Insulated Gate Field-Effect Transistor Switching Circuits," *IEEE Journ. on Solid State Circuits* Vol. SC-**3** pp. 285-289 (Sept. 1968).

24. L.O. Chua and P.M. Lin, *Computer-Aided Analysis of Electronic Circuits: Algorithms & Computational Techniques*, Prentice-Hall, Inc., Englewood Cliffs, N.J. (1975).

25. J.L. Burns, A.R. Newton, and D.O. Pederson, "Active Device Table Look-up Models For Circuit Simulation," *Proc. 1983 Int. Symp. on Circ and Sys.*, (May 1983).

26. Jiri Vlach and Kishore Singhal, *Computer Methods for Circuit Analysis and Design*, Van Nostrand Reinhold Publishing, New York, N.Y. (1983).

27. J.M. Ortega and W.C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York (1970).

28. G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, Maryland (1983).

29. A.R. Newton and A.L. Sangiovanni-Vincentelli, "Relaxation-Based Electrical Simulation," *IEEE Trans. on Electron Devices*, pp. 1184-1207 (Sept. 1983).

30. B.R. Chawla, H.K. Gummel , and P. Kozak, "MOTIS - An MOS timing simulator," *IEEE Trans. on Circ. and Sys.* **CAS-22** pp. 901-909 (Dec. 1975).

31. A.R. Newton, "The Simulation of Large-Scale Integrated Circuits," *Memo UCB/ERL M78/52*, University of California, (July 1978). Ph.D. Dissertation.

32. J. White, "RELAX2 : A Modified Waveform Relaxation Approach To The Simulation of MOS Digital Circuits," *Memo UCB/ERL M84/21*, University of California, (22 Feb 1984).

33. J. White, F. Odeh, A. S. Vincentelli, and A. Ruehli, *Waveform Relaxation: Theory and Practice*.

34. A. R. Newton and A. S. Vincentelli, *EECS 244 Lecture Notes*, University of California (SP/1985).

35. C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Problems*, Prentice-Hall, Inc, Englewood Cliffs, NJ (1971).

36. G. Micheli, A.R. Newton, and A. S. Vincentelli, "Symmetric Displacement Algorithms for the Timing Analysis of Large Scale Circuits," *IEEE. Trans. on CAD, Vol. CAD-2, No.3*, (July 1983).

37. A.R. Newton, "The Analysis of Floating Capacitors for Timing Simulation," *Proc. 13th Asilomar Conference on Circuits Systems and Computers*, (November 1979).

38. Tammy Huang, *Generalization of the Implicit-Implicit-Explicit method for Floating Capacitors*, University of California, Berkeley (1983).

39. A. Richard Newton, "Techniques for the Simulation of Large-Scale Integrated Circuits," *IEEE Trans. on Circ. and Sys.* **CAS-26, NO. 9** pp. 741-749 (Sept. 1979).

40. Jacob White, Resve A. Saleh, A. Sangiovanni-Vincentelli, and A. R. Richard Newton, "Accelerating Relaxation Algorithms for Circuit Simulation," *Digest 1985 Int. Conf. on CAD, IEEE*, (Nov 1985).

41. Guy Marong and A. Sangiovanni-Vincentelli, "Waveform Relaxation and Dynamic Partition for the Transient Simulation of Large Scale Bipolar Circuits," *Digest 1985 Int. Conf. on CAD, IEEE*, (Nov 1985).

42. Hitchcock, "Timing Verification and the Timing Analysis Program," *19th Design Automation Conference,IEEE*, pp. 594-604 (June 1982).

43. J.K. Ousterhout, "Crystal:A Timing Analyzer for NMOS VLSI Circuits," *Proceeding of the third Caltech VLSI Conference*, pp. 57-59 (1983).

44. N. P. Jouppi, "Timing Analysis for NMOS VLSI," *Proceedings of the 20th Design Automation Conference*, pp. 411-418 (1983).

45. D. J. Pilling and J. C. Skalnik, "A Circuit Model for Predicting Transient Delays in LSI Logic Systems," *Proc. 6th Asilomar Conference on Circuit and Systems,* pp. 424-428 (1972).

46. Christopher C. Marino, *Smalltalk on a RISC - CMOS Implementation,* University of California, Berkeley (May). M.S. Report

47. Pauline Ng, Wolfram Glauert, and Robert Kirk, "A Timing Verification System Based on Extracted MOS/VLSI Circuit Parameters," *18th Design Automation Conference,IEEE,* pp. 288-192 (June 1981).

48. McWilliams, T.M., "Verification of Timing Constraints on Large Digital Systems," *17th Design Automation Conference,IEEE,* pp. 139-147 (1980).

49. T.I. Kirkpatrick and N.R. Clark, "PERT as an Aid to Logic Design," *IBM Jour. of Research and Development,* pp. 135-141 (1966).