ITERATED TIMING ANALYSIS AND SPLICE1

by

R. A. Saleh

Memorandum No. UCB/ERL M84/2

4 January 1984

ITERATED TIMING ANALYSIS AND SPLICE1

by

Resve A. Saleh

Memorandum No. UCB/ERL M84/2

4 January 1984

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

SPLICE1 is a mixed-mode simulation program for large-scale integrated circuits. It performs concurrent electrical and logic simulation using event-driven selective-trace techniques. The electrical analysis uses a new algorithm, called Iterated Timing Analysis (ITA), which performs accurate electrical waveform analysis much faster than SPICE2 for large circuits. The logic analysis features a new MOS-oriented state model and a fanout dependent delay model, and handles bidirectional transfer gates in a consistent manner.

This report describes the new algorithms and the details of the implementation in SPLICE1.7. Program performance characteristics and a number of simulation results are also included.

Acknowledgements

I would like to express my appreciation to my research advisor Prof. A. Richard Newton for his patience, encouragement and guidance throughout course of this work. I would also like to thank Prof. Don O. Pederson and Prof. Alberto Sangiovanni-Vincentelli for their support.

I wish to thank everyone in the CAD group at Berkeley but a few people deserve special mention. In particular, I would like to thank Jim Kleckner for the many long hours of help generating examples, fixing bugs and engaging in useful discussions. I would also like to thank Jacob White for the discussions on the theoretical aspects of the work. I am grateful to Ben Valdez of Hughes Aircraft for reviewing the manuscript carefully, providing useful suggestions for program development and for assistance with examples.

Graeme Boyle, John Crawford, Ian Getreu, Jack Hurt and Steve Potter of Tektronix helped me a great deal during the course of the project. I would also like to express my special thanks to Mike Caughey and the ICCAD group at MITEL Corp. in Ottawa, Canada for their encouragement.

A number of designers have helped during the initial debugging phase of this work. In particular, I would like to thank Don Herbert of Aerospace Corporation, Professor Miles Copeland of Carleton University, Ottawa, Canada, Ron Jerdonek of the General Electric Co., and Marcus Paltridge, Chris Wilson, and Craig Mudge of CSIRO VLSI, Australia.

Both the Digital Equipment Corporation and Toshiba Corporation provided

state-of-the-art test circuits and help with debugging. In particular, I would like to thank Ed Burdick of Digital and Takayasu Sakurai of Toshiba Corporation for their help.

Finally, I wish to thank my wife, Lynn, and my family members in Ottawa, Canada for their continuing support.

This work was supported in part by NSERC (Natural Science and Engineering Research Council) of Canada, the Hewlett-Packard Company, Digital Equipment Corporation and Tektronix.

TABLE OF CONTENTS

CHAPTER 1

1. INTRODUCTION

SPLICE1 is a *mixed-mode* simulation program for large digital MOS integrated circuits (IC). It performs time-domain transient analysis which tends to be the most time-consuming and memory-intensive task in simulation today. The enhancements made to the program are described in this report. The starting point for this work was SPLICE1.3 [1]. This early version of SPLICE1 included 4-state logic simulation, simple timing analysis and a SPICE-like circuit simulation capability [1, 2].

While this version provided a degree of functionality, it suffered from modeling and accuracy problems intrinsic to the algorithms used in the program. Specifically, the 4-state logic model was not sufficient to perform an accurate true-value logic simulation of general MOS circuits containing transfer gates and wired connections (i.e., more than one gate controlling the state of a node). The simple timing analysis algorithm had inherent accuracy limitations and stability problems and had difficulty analyzing circuits containing floating elements and tight feedback loops. These, and other issues, are examined in detail elsewhere [3]. and will be elaborated further in later sections.

The latest version, SPLICE1.7, overcomes these problems by using state-of-the-art algorithms in place of previous ones.

The electrical analysis is performed using a new technique called *Iterated Timing Analysis* (ITA) which can be derived from simple timing analysis [4, 5]. In this approach, the set of nonlinear circuit equations are solved using a relaxation-based method rather than a method which requires the direct solution of a set of linear equations, usually found in standard

circuit simulators such as SPICE2 [6]. ITA is as accurate as SPICE2, assuming identical device models, and has guaranteed convergence and stability properties. Due to the selective trace feature in SPLICE1, the execution time can be up to two orders of magnitude faster than SPICE2, with comparable waveform accuracy, for large circuits. Another key feature of ITA is its ability to perform accurate analysis of complex analog circuits, as will be shown later. Iterated Timing Analysis has shown so much promise that efforts are being directed to generalize it as a standard technique for accurate electrical simulation. Therefore, a matrix-oriented simulation capability is no longer available in SPLICE1.

The logic analysis capabilities have also been extended to include the notion of multiple strengths or impedance levels [3] as is available in most modern MOS-oriented logic simulators [7, 8, 9, 10]. While other simulators usually limit the number of strengths to three, there is practically no limit in SPLICE1.7, which allows up to 2^{16} - 1 strengths. More than three strengths are often required to model the interaction between transfer gates of differing geometry [3]. Processing of the gates and nodes proceeds in a manner similar to the electrical analysis. In fact, the logic analysis may be thought of as a relaxation-based method in which the elements are represented by simple logic models rather than complex analytical equations. This concept, together with the idea of multiple impedance levels, allows for a more consistent signal representation and signal conversion in the mixed-mode environment. Clearly, there is a correspondence between an electrical voltage and the logic levels. With the notion of strengths, there is now a natural correspondence between the electrical output conductance of an element and the logic output strength of the element.

SPLICE1 can also be used to perform switch-level simulation [9, 10] to verify circuit functionality at the transistor level. It handles CMOS, NMOS and PMOS circuits in both static and dynamic configurations.

Although SPLICE1 originally included a table look-up scheme to speed up MOS model evaluation [4, 5], it was subsequently dropped from the program. Research on optimal table models and structures is continuing in an independent effort [11] and this feature may be reinstated in a later version. Therefore, this report does not address the issue of table-driven MOS models.

The remainder of this report is divided into four chapters. In Chap. 2, the ITA algorithm is described in detail. The enhancements in the logic analysis are described in Chap. 3. In Chap. 4, a number of simulation results and program performance statistics are presented. Finally, in Chapter 5, the general conclusions are stated with specific mention of future directions.

CHAPTER 2

2. Iterated Timing Analysis

2.1. Introduction

A new form of electrical analysis, called *Iterated Timing Analysis* (ITA), is described in this chapter. The motivation for this work is presented using SPLICE1.3 as an example of Non-iterated Timing Analysis (NTA). A simple mathematical treatment of the ITA method is presented here although a complete mathematical analysis of relaxation-based methods, presented in a rigorous and unified framework, may be found in reference [12]. The details of the implementation in SPLICE1.7 are also included in this chapter.

2.2. The Simulation Problem

The general circuit analysis problem in the time domain requires the solution of a set of first-order nonlinear Ordinary Differential Equations (ODE) of the form:

$$C(v(t),u(t))\dot{v} = -f(v(t),u(t)) \qquad (2\text{-}1)$$

where
$v(t)$ is the set of unknown node voltages,
$u(t)$ is the set of inputs,
$C(v(t),u(t))$ is the nodal capacitance matrix,
$f(v(t),u(t))$ is the sum of the currents charging the capacitances at each node.

This formulation can be derived by writing *Kirchoff's Current Law* (KCL) at every node, except the ground node, in a given circuit [12]. The simulation task is to determine the unknown voltages, $v(t)$, for every node at every

timepoint due to some input excitation, $u(t)$.

The technique used in SPICE2 to solve Eqn. (2-1) is to first convert the set of differential equations into a set of algebraic difference equations using a stiffly-stable integration formula [6]. The nonlinear difference equations are then converted to a set of linear equations of the form:

$$GV = I \tag{2-2}$$

using a damped Newton-Raphson linearization process. G is the Jacobian matrix (or the small-signal conductance matrix), V is the unknown voltage vector and I is the known excitation vector. Next, Eqn. (2-2) is solved using a direct matrix approach to produce the solution vector, V. Since, in general, $f(v(t),u(t))$ is a nonlinear function, this process must be repeated until V converges to a consistent solution.

2.3. Motivation for a New Simulation Approach

General-purpose simulation programs, such as SPICE2 [6] and ASTAP [13], have been used extensively to perform accurate circuit analysis for over 10 years. These simulators use direct methods (using sparse matrix techniques) to solve the set of circuit equations. Unfortunately, this approach becomes increasingly expensive as the circuit size increases. The fundamental problem is illustrated in Fig. 2.1. The time required to formulate the set of linear equations grows linearly with circuit size whereas the time required to solve the linear equations is proportional to N^k, where N is the number of circuit nodes and k ranges from 1.1 to 1.5. These two solution phases are referred to in Fig. 2.1 as FORM and SOLVE respectively. The SOLVE phase quickly dominates the total time as the circuit size increases

and this is one reason why the direct approach is not appropriate for large circuits.

Timing simulation was introduced in the mid-seventies to reduce CPU-time at the expense of some accuracy. A new breed of simulators emerged at that time, all tailored to perform transient analysis of large digital circuits [5, 4, 1, 14, 15]. These classical timing analysis programs used iterative techniques [16] to solve the set of circuit equations rather than the direct matrix solution approach of the previous generation. A grounded capacitor was required at every node to guarantee convergence of the method. However, to reduce execution time, none of these programs carried the iteration to convergence and in fact each node equation was solved once at each timepoint.

Large digital circuits typically display a 10-20 % *latency* characteristic. That is, only 10-20 % of the nodes in the circuit are active at any given time. Conceptually, there is temporal sparsity (latency in a waveform over a time period) and spatial sparsity (latency in the network at a given point in time) [17]. Since these iteration methods involve the solution of each equation separately, this latency aspect can be exploited to further improve performance.

Using these techniques, two orders of magnitude of speed improvement was obtained. Accuracy was maintained in these simulators by choosing a small fixed timestep for the entire analysis. This timestep was either constant for all circuits [5] or chosen based on the smallest time constant in the circuit [4]. Later timing simulators adjusted the timestep during the analysis dynamically to limit the voltage change over a timestep to a value specified by the user [1]. Simple timing analysis also relied heavily on the

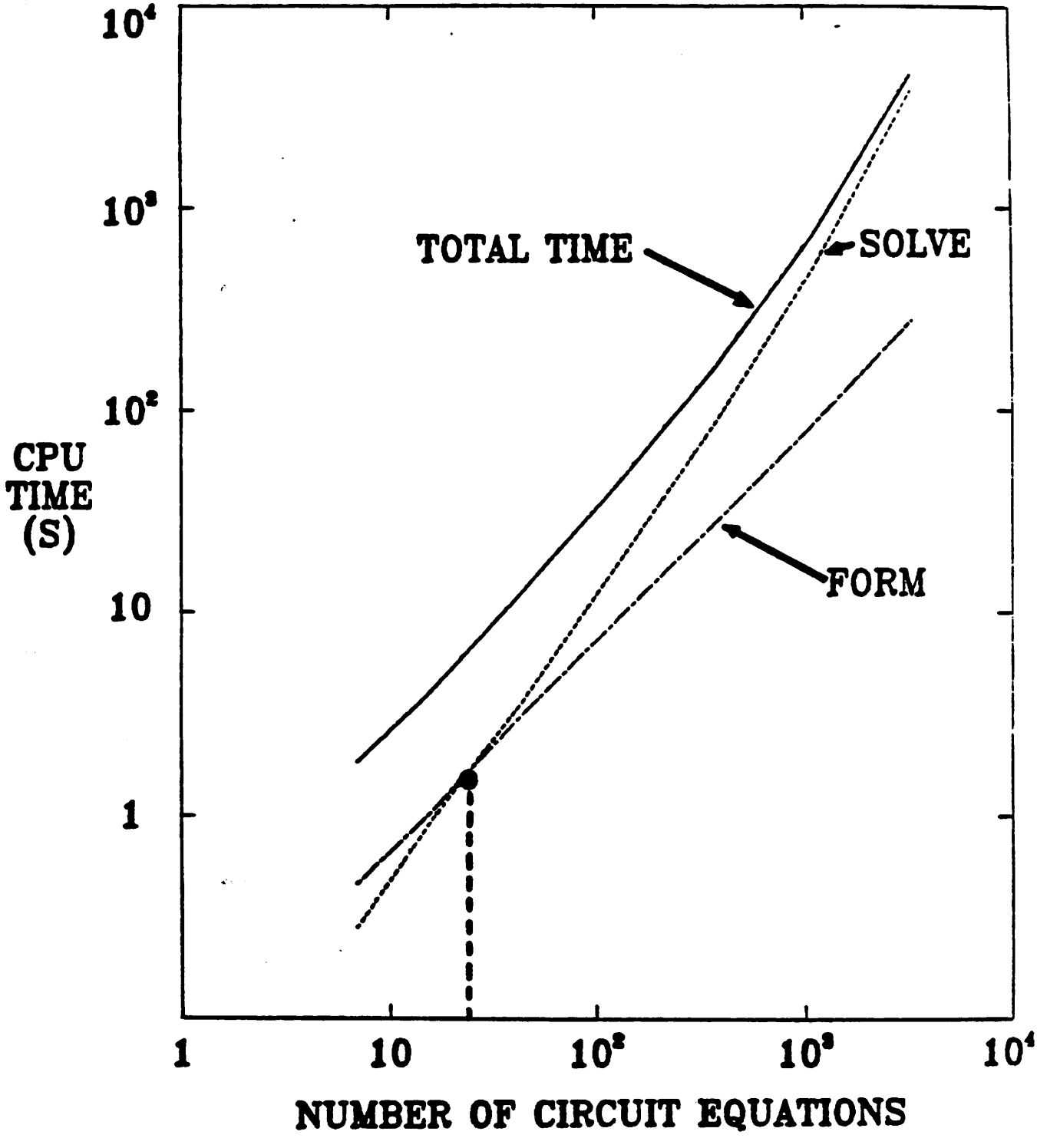


Fig. 2.1 : the amount of CPU time required to perform a transient analysis of a set of typical circuits of increasing size

fact that the accumulated voltage error becomes zero once a node reaches either of the two supply voltages.

While offering a substantial savings in CPU-time and memory usage, these programs suffer from a number of problems which have limited their use. Circuits containing global feedback loops, such as the ring oscillator of Fig 2.2(a), produce timing and voltage errors in the simulation. Fig 2.2(b) illustrates these errors as generated by SPLICE1.3 compared to the solution produced by SPICE2. The SPLICE1.3 program not only produces a timing error (phase error) but incorrectly predicts the number of cycles in a given time period (frequency error) and the height of the peaks (amplitude error).

These errors are all due to the single iteration performed at each timepoint. To understand the origin of the errors, consider the processing sequence of a simple NMOS inverter of Fig. 2.3(a) using NTA.

(1) The first step is to represent each nonlinear device by its corresponding linear companion model. This is done in Fig. 2.3(b). These equivalent models are based on the terminal voltages of each device. The conductance is obtained from the slope of the nonlinear I-V characteristic at the operating point and the current is obtained from the y-axis intercept as shown in Fig. 2.3(c). The value of the voltage at Node C is calculated using this equivalent circuit of Fig. 2.3(b). Assume that initially $V_B^{n-1}=0v$ and $V_C^{n-1}=5.0v$, where V_B^{n-1} refers to the voltage at node B at time t_{n-1} and V_C^{n-1} has a similar definition.

(2) Let $V_B^n=1.0v$. Then the change in the voltage at node C is calculated using V_B^n and V_C^{n-1}. Therefore, the linear equivalent model of the load is the same as it was at t_{n-1} but the equivalent model of the driver changes. Since the load offers less charging current than it really

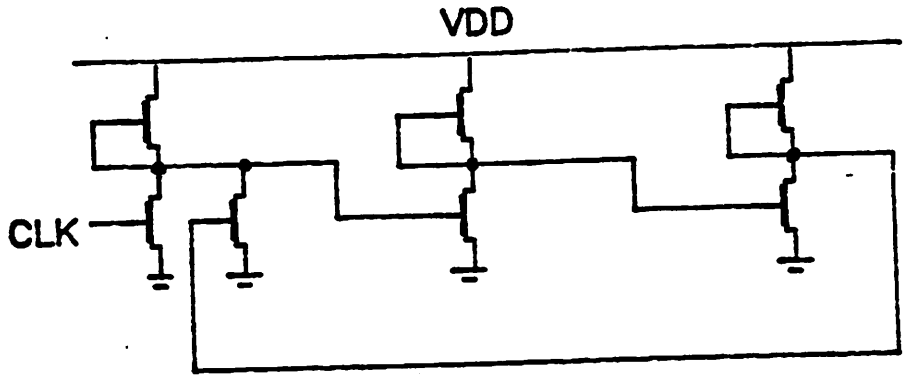


Fig. 2.2(a) : NMOS Ring Oscillator Circuit

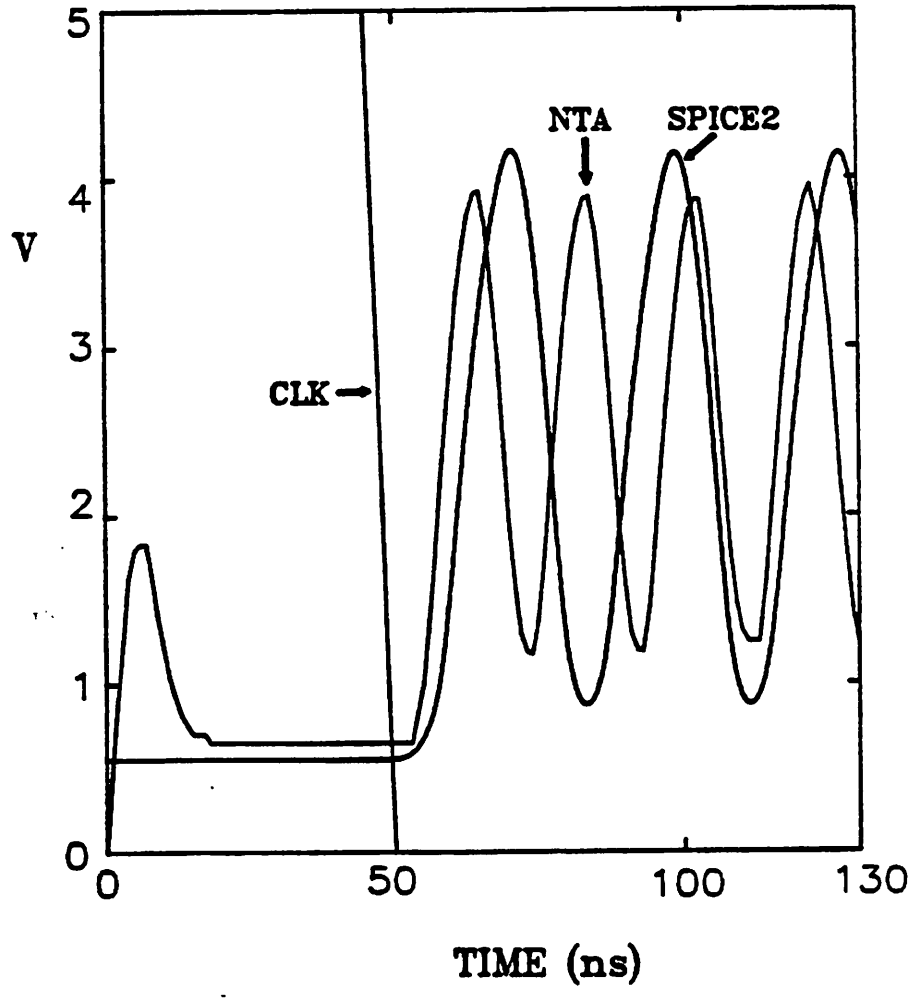


Fig. 2.2(b) : Comparison of NTA and SPICE2. The errors are due to the single iteration performed at each timepoint.

should (i.e., V_C is incorrect) and the driver is able to sink more current due to a larger V_{ds}, the node voltage change ΔV_C^n is too optimistic.

(3) When $V_B^{n+1}=2.0v$, again $V_C^{n+1}=f(V_C^n, V_B^{n+1})$ and ΔV_C^{n+1} is also optimistic by the same argument given above.

Hence, in a SPLICE1.3 simulation, the output of the inverter will rise and fall earlier in time with faster rise and fall times than the SPICE2 simulation of the same circuit. This error is propagated and intensified in the ring oscillator circuit, resulting in the three errors cited above. It should be noted that if the timestep of the simulation is reduced and the accuracy tolerances are tight, the NTA output will be indistinguishable from SPICE2 output for this example.

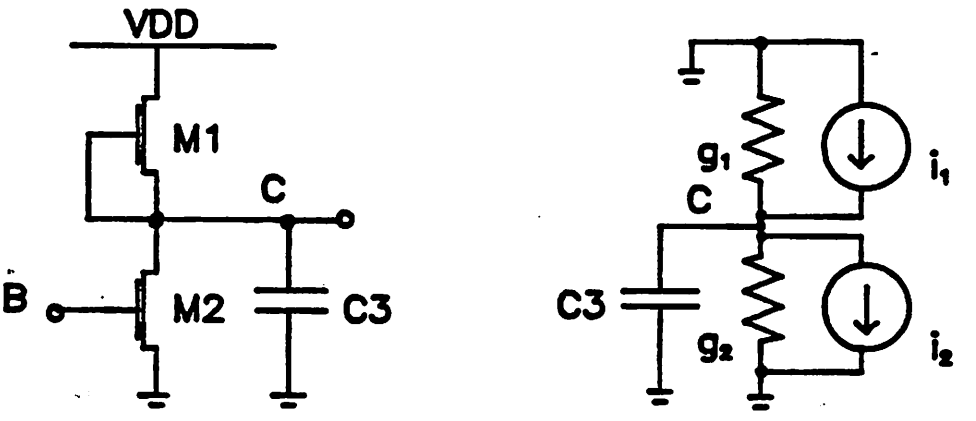
Another shortcoming of NTA is that it has some difficulty dealing with circuits containing floating elements, such as capacitors and transfer gates. These elements introduce strong bilateral coupling between two nodes in the circuit. Since only one iteration is used, the solution obtained using NTA depends on the order that the nodes are processed. Consider, for example, the 2-input NMOS NAND circuit of Fig. 2.4(a). It contains a "floating" transistor, namely M2. The sequence of processing in the NTA method would be as follows :

(1) Assume that initially $V_X^{n-1}=0v$, $V_Y^{n-1}=5.0v$ and $V_B^{n-1}=0v$

(2) At t_n, $V_B^n=1.0v$ and Node X is processed using the initial conditions V_X^{n-1} and V_Y^{n-1} given above to produce V_X^n.

(3) Next, Node Y is processed using V_Y^{n-1} and V_X^n to generate V_Y^n.

(4) Then, time is advanced by one unit and steps (2) to (4) are repeated. This process continues until Node Y makes its transition to the opposite rail voltage.



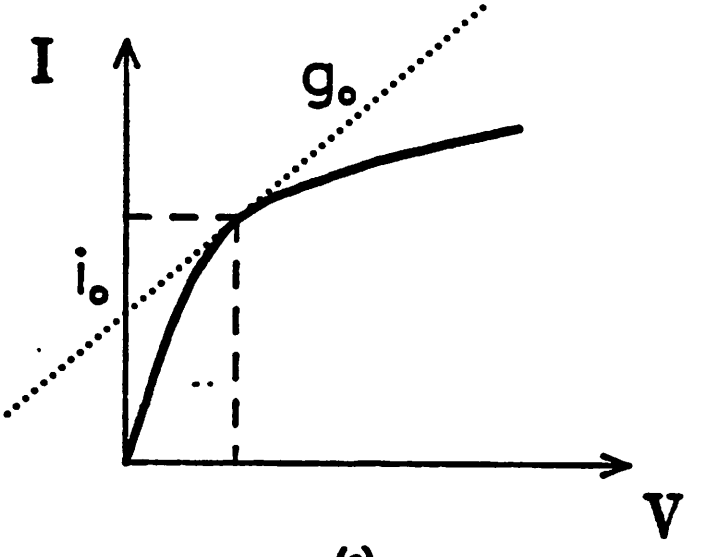


Fig. 2.3 : The linear equivalent model for each transistor of (a) is shown in (b) using the device characteristics in (c)

There are two problems with this method:

- the change in node Y should immediately affect node X but it is not reflected at node X until the next time point.
- if the processing started with node Y instead of node X, slightly different results would be obtained.

The same effect is observed when processing capacitors where one node is not connected to ground (i.e., a floating capacitor). An example of a circuit with such a capacitor is the boot-strapped inverter of Fig 2.4(b). The accuracy of NTA depends on the timestep and the ratio of the floating capacitor to the grounded capacitor. In the boot-strapped inverter, the value of C03 is usually large compared to the grounded capacitors, C01 and C02, and this tends to reduce the accuracy of the solution produced by NTA.

Therefore, NTA will produce somewhat inaccurate results when there are floating elements in the circuit. Reducing the timestep to an appropriate value will improve the accuracy, but if the timestep is not small enough, these elements may cause the simulator to exhibit instability. As will be seen later in this section, the ITA approach overcomes all of these problems.

By far the most compromising aspect of the NTA approach is that it may occasionally produce the wrong answer! Circuit designers are willing to use a program which gives them the correct answer or no answer (usually due to non-convergence), but are unable to deal with a program that occasionally produces the wrong answer. In fact, the NTA method *always* produces some answer and this is really the downfall of the method.

For the reasons given above, timing analysis has not been widely accepted as viable form of electrical simulation, although it has been used successfully in constrained IC design methodologies such as standard cell or

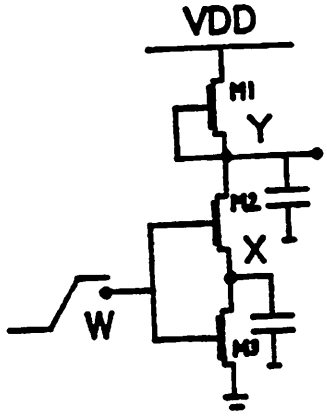


Fig. 2.4(a) : 2-input NMOS NAND circuit. The floating transistor connected between nodes X, Y, and W causes problems for NTA.

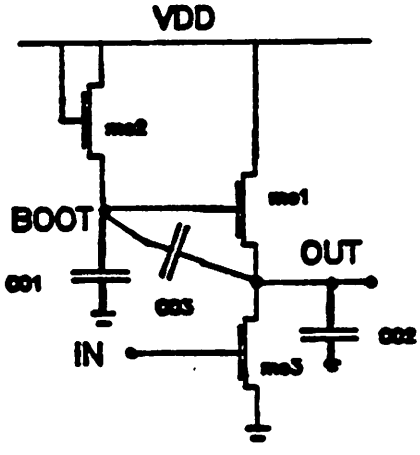


Fig. 2.4(b) : Boot-strapped Inverter. The capacitor, COS, causes problems for NTA.

gate array. What is really required is a simulation technique which provides both accuracy and speed.

2.4. Relaxation-based Electrical Simulation

A number of new techniques have been developed in an effort to reduce the simulation time while maintaining waveform accuracy comparable to SPICE2. These include table-driven model evaluation [11], microcode tailoring on a minicomputer [18] and the use of vector-oriented computers such as the CRAY-1 [19]. Although these techniques have been successful, they provide, at most, an order of magnitude speed improvement over SPICE2.

Two methods are currently being investigated which use a converged relaxation iteration to solve the set of circuit equations. Both approaches have been implemented and preliminary results indicate that up to two orders of magnitude of speed improvement may be obtained for large digital circuits. One method, called *Waveform Relaxation*[20], decomposes the system of equations into several dynamic subsystems each of which is analyzed for the entire simulation period. The process is then repeated until all the waveforms converge to an exact solution. The relaxation is performed at the differential equation level. This method has been implemented in program RELAX [20, 21].

The second method is called *Iterated Timing Analysis* (ITA) [22, 23]. In this method, the relaxation is performed at the nonlinear equation level. That is, the set of *nonlinear* circuit equations are iterated to convergence using a Gauss-Seidel or Gauss-Jacobi method. This is also an exact method. Some aspects of this method which make it attractive are as follows:

- it has guaranteed convergence and stability properties
- it allows circuit latency to be exploited easily
- it can be implemented using the concepts developed for logic simulation
- since the logic and electrical analyses operate the same way, a consistent mixed-mode simulation is possible

The algorithm has been implemented in SPLICE1.7 and the implementation details and results obtained are presented in this chapter following a simple mathematical treatment of the method.

2.5. The ITA Algorithm

2.5.1. The Gauss-Seidel Iteration Method

A system of simultaneous *linear* equations can be solved using a variety of techniques, namely:

1. Direct Methods
   a. Matrix Inversion
   b. Gaussian Elimination
   c. LU decomposition

2. Iterative Methods
   a. Gauss-Jacobi
   b. Gauss-Seidel

In circuit simulation, the solution to Eqn. (2-2) is required. The circuit conductance matrix, G, is usually large but sparse, typically having 3 elements per row. Matrix inversion is not a suitable method because it usually converts a sparse matrix into a dense one. Sparse matrix techniques can be used to solve the equations using method 1(b) or 1(c) but this is not suitable for large circuits due to the rapid increase in CPU-time, as shown in Fig. 2.1.

The iterative methods [16] are well-suited to cases where the matrix is sparse. In fact, the solution of a set of sparse linear equations may be obtained faster using an iterative approach. Two classical iteration methods exist: the Gauss-Jacobi (G-J) method and the Gauss-Seidel (G-S) method. The Gauss-Jacobi method (also referred to in the literature as simultaneous displacement) proposes the following approach:

$v^{(0)}$ = initial guess voltage vector
$m \leftarrow 0$
repeat {
    for (i = 1 to N) {

$$v_i^{m+1} = \frac{1}{g_{ii}}\left[i_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} g_{ij} v_j^{m}\right] \tag{2-3}$$

    }
    $m \leftarrow m+1$
} until $|v_i^{m+1} - v_i^{m}| \leq \varepsilon$ for all i, i.e., convergence

Notice that every equation uses the *previous* iteration values for all unknown voltages to obtain a new solution vector. The Gauss-Seidel method (also referred to as successive displacement) suggests the following modification to Gauss-Jacobi:

$v^{(0)}$ = initial guess voltage vector
$m \leftarrow 0$
repeat {
    for (i = 1 to N) {

$$v_i^{m+1} = \frac{1}{g_{ii}}\left[i_i - \sum_{j=1}^{i-1} g_{ij} v_j^{m+1} - \sum_{j=i+1}^{n} g_{ij} v_j^{m}\right] \tag{2-4}$$

    }
    $m \leftarrow m+1$
} until $|v_i^{m+1} - v_i^{m}| \leq \varepsilon$ for all i, i.e., convergence

Notice that each equation uses the *latest* values of voltage wherever possible.

The only difference in the two methods is whether the previous voltages are always used or the latest values are applied immediately. The convergence rate is linear in both cases but the speed of convergence is quite different. Usually the Gauss-Seidel iteration converges faster than Gauss-Jacobi [16], although there are cases where this is not true.

Both methods also require the strict diagonal dominance condition for guaranteed convergence:

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} |g_{ij}| < |g_{ii}| \tag{2-5}$$

This inequality states that each diagonal term of the matrix be greater than the sum of all the off-diagonal terms in the same row.

An acceleration scheme is available to speed up convergence using an acceleration parameter, ω, as follows.

$$v_i^{m+1} = \omega v_i + (1-\omega) v_i^m \tag{2-6}$$

where v_i is an intermediate value generated using Eqn. (2-4). The effect of ω is usually dramatic but it can only be obtained empirically and usually varies from technology to technology. For standard Gauss-Seidel, $\omega = 1$.

2.5.2. A Nonlinear Gauss-Seidel Iterative Approach

Relaxation methods, as described in the previous section, can also be applied successfully at the nonlinear equation level. The same approach is used as for linear equations except that each nonlinear equation must first be linearized and solved before proceeding to the next equation. Using this approach, the time-consuming effort required to calculate the Jacobian matrix entries can be avoided.

The steps at the nonlinear equation level are as follows. Starting with equation (2-1), the first step is to convert the differential equations into difference equations using a stiffly-stable integration formula. SPLICE1.7 uses a Backward-Euler formulation [24]. Then the first equation is linearized using the Newton-Raphson (N-R) method and iterated to convergence to solve for one unknown voltage. This constitutes the inner N-R loop. The same process is applied to the next equation and all subsequent equations, in turn, until the last equation is processed. This outer G-S loop is now iterated to convergence to produce the solution.

To further illustrate the method, consider the solution method applied to one node in a typical circuit. Fig 2.5(a) shows three nonlinear devices connected to Node 4, which has a capacitor connected to ground. We begin by writing KCL for Node 4

$$\sum_{j=1}^{4} I_j = I_4+I_1+I_2+I_3 = 0 \tag{2-7}$$

This can be rewritten in the form of Eqn. (2-1) :

$$C_4\dot{V}_4 = -(I_1(V_1,V_4) + I_2(V_2,V_4) + I_3(V_3,V_4)) \tag{2-8}$$

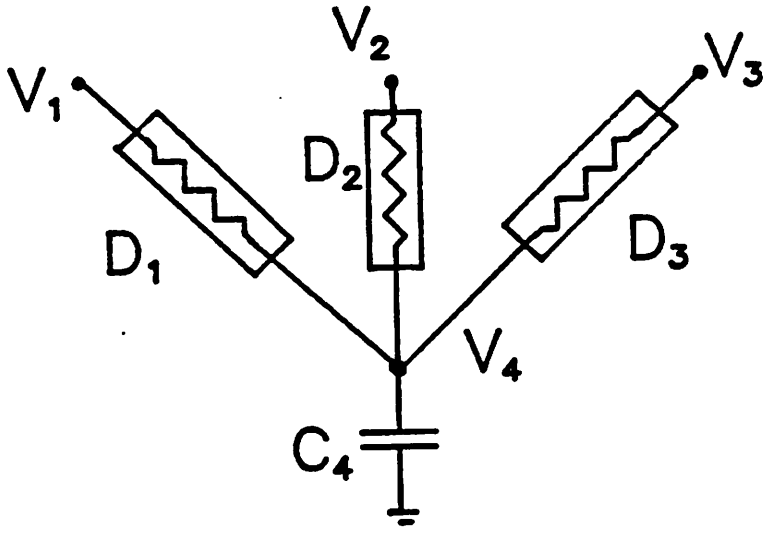
Using the Backward-Euler formula for I_4 , we obtain

$$I_4 = C_4\frac{dV_4}{dt} = \frac{C_4}{h}(V_{4(n)}-V^{*}_{4(n-1)}) \tag{2-9}$$

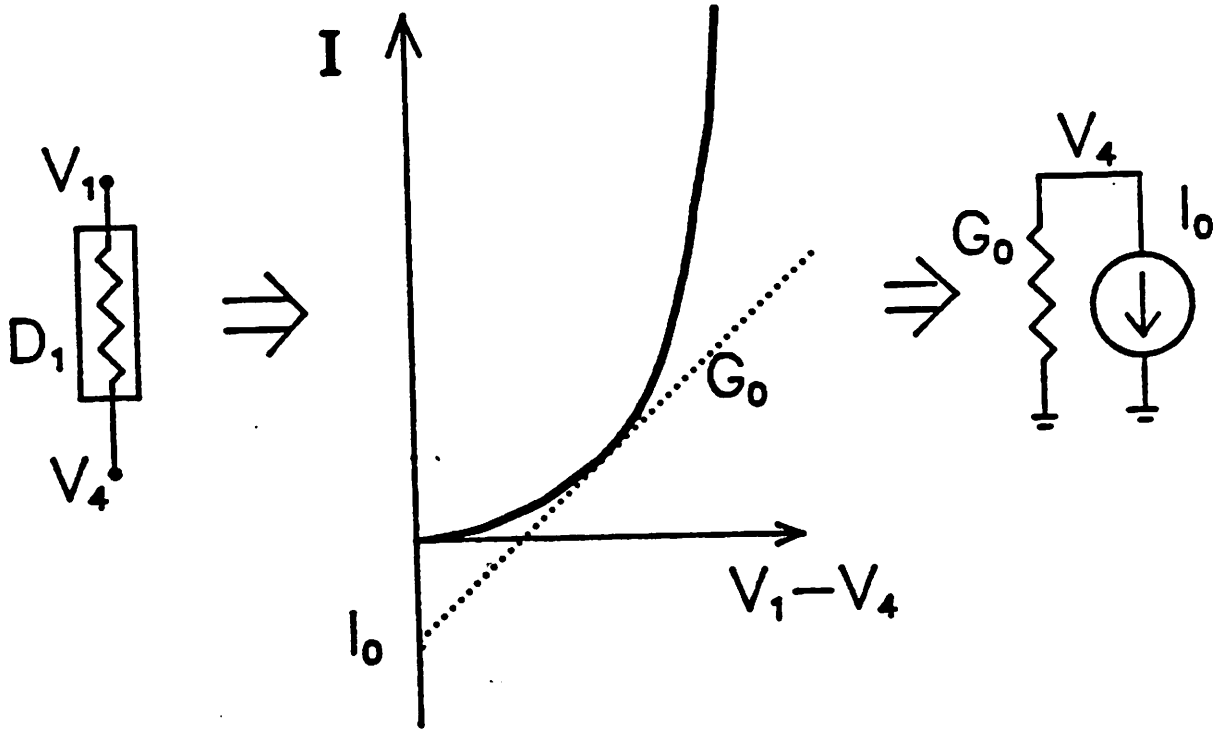
where h is the integration step size, $V_{4(n)}$ refers to the voltage value for Node 4 at time t_n and $V^{*}_{4(n-1)}$ refers to the solution obtained for Node 4 at time t_{n-1}. Therefore, Eqn. (2-8) can now be written as a difference equation,

$$\frac{C_4}{h}(V_{4(n)}-V^{*}_{4(n-1)}) + I_1(V_1,V_4) + I_2(V_2,V_4) + I_3(V_3,V_4) = 0 \tag{2-10}$$

Since Eqn. (2-10) has the form:



(a)



(b)

Fig. 2.5 : The equation used in the Nonlinear Gauss-Seidel iteration is derived using circuit (a). The companion model for each nonlinear device is obtained using the process shown in (b).

$$f(V_1, V_2, V_3, V_4) = 0 \tag{2-11}$$

it is suitable for the Newton-Raphson (N-R) iterative method with V_4 as the unknown variable. The general equation for one N-R iteration is

$$x^{(i+1)} = x^{(i)} - \frac{f(x^{(i)})}{f'(x^{(i)})} \tag{2-12}$$

In circuit terms, the N-R calculation usually requires that a linear equivalent be determined for each nonlinear device connected to the node, as shown in Fig. 2.5(b) for D1. This involves the calculation of a conductance, G_0 and a current intercept, I_0. In order to avoid the intercept calculation, we can apply Eqn. (2-11) directly to Eqn. (2-12) to get

$$V_{4(n)}{}^{i+1} = V_{4(n)}{}^{i} - \frac{f(V_1, V_2, V_3, V_4)}{f'(V_1, V_2, V_3, V_4)} \tag{2-13}$$

Now set $\Delta V_{4(n)}{}^{i+1} = V_{4(n)}{}^{i+1} - V_{4(n)}^{i}$ and substitute Eqn. (2-10) into Eqn. (2-13) to get

$$\Delta V_{4(n)}{}^{i+1} = \frac{\sum_{j=1}^{3} I_j^i + \frac{C_4}{h}(V_{4(n)}{}^{i} - V_{4(n-1)}^{*})}{\sum_{j=1}^{3} G_j^i + \frac{C_4}{h}} \tag{2-14}$$

where $V_{4(n)}{}^{i}$ refers to the ith iteration value of voltage at Node 4 at time t_n and I_j^i refers to the ith iteration value of current at Node j.. This method of evaluating ΔV is convenient because:

- no intercepts need to be calculated since total currents are used in Eqn. (2-14)
- current levels are within operating ranges (unlike I_0 in Fig. 2.5(b))
- the value of ΔV is very accurate when calculated this way. Note that ΔV is the difference between two Newton iterations and it will tend toward

zero with each iteration. Therefore it should be calculated as accurately as possible.

For an arbitrary node Eqn. (2-14) becomes

$$\Delta V^{i+1} = \frac{\sum_j^n I_j^i + \frac{C}{h}(V_n^i - V_{(n-1)}^*)}{\sum_j^n G_j^i + \frac{C}{h}} \qquad (2\text{-}15)$$

2.5.3. The SOR-Newton Iteration

A combination of the Newton-Raphson iteration in a converged Gauss-Seidel loop with acceleration applied is called the SOR-Newton method. In equation form, it is simply

$$\Delta V = -\frac{\omega f(V)}{f'(V)} \qquad (2\text{-}16)$$

In a standard N-R iteration, the equation is iterated until $|\Delta V| \leq \varepsilon$. This means that each node equation should be iterated to convergence before moving on to the next one. The Gauss-Seidel loop (i.e., the outer loop) must also be iterated to convergence.

2.5.4. Convergence of the SOR-Newton Iteration

A very important property of the SOR-Newton iteration can be applied now to greatly reduce the number of iterations of the inner N-R loop. It happens that *one* Newton iteration per equation for each G-S iteration is sufficient to retain the convergence properties of the nonlinear Gauss-Seidel iteration [16] as long as the convergence requirements of the N-R iteration are strictly satisfied.

A Newton-Raphson iteration will converge if the initial guess is "close enough" to the exact solution, given that the function is Lipschitz

continuous. Under these conditions, the rate of convergence is quadratic. Since the element model equations are smooth, the solution from one timepoint to the next will not be drastically different. Therefore, the solution at the previous timepoint is a good first guess for the N-R iteration. Furthermore, a prediction step may be used to generate a better first guess. A simple linear predictor is used in SPLICE1 using the previous two solution points.

The diagonal dominance requirement for the G-S iteration must also be satisfied to guarantee the convergence of the SOR-Newton iteration. In circuit terms, this requirement can always be met by placing a grounded capacitor at every node and choosing an appropriate timestep. Grounded capacitors appear as $\frac{C}{h}$ terms in the diagonal position of the conductance matrix G. Therefore, h, which is the simulation timestep, can be reduced until the $\frac{C}{h}$ term is greater than the sum of all off-diagonal terms.

Off-diagonal terms appear in the conductance matrix when there is coupling between two nodes. For example, when floating capacitors are used, $\frac{C}{h}$ terms appear in diagonal and off-diagonal positions. Therefore, reducing the value of h is not as effective and this may lead to convergence problems. The ratio of the floating capacitor to the grounded capacitor is an important factor in determining the speed at which convergence is achieved. If the floating capacitor is very large compared to the grounded capacitor, convergence speed will be slow, if the iteration converges at all. The current version of SPLICE uses the IIE method (Implicit-Implicit-Explicit) [25] to evaluate floating capacitors.

2.6. Exploiting Latency

SPLICE1 does not solve every node at every timepoint. In fact, only those nodes which are active at any given point in time are processed. Since large circuits are relatively inactive, less than 20% of the nodes are actually solved at each timepoint. The active nodes are determined on an event-driven basis. That is, a node is placed in the set of active nodes if any node which can affect it changes by a significant amount.

Once the set of active nodes are identified, SPLICE1 can exploit two forms of latency. The first one is called simply *latency in time*. This is based on the fact that digital circuit waveforms feature long constant periods. An active node is processed at consecutive points in time until it reaches a constant value. It is then removed from the set of active nodes and becomes latent. The second form of latency is the so-called *latency at a timepoint*. This refers of the fact that some nodes may actually converge with fewer iterations than others, at a given timepoint. These nodes can be marked to be processed at the next timepoint while the remaining nodes continue to iterate to convergence at the current timepoint. Tightly-coupled nodes usually require more iterations than other nodes.

The decoupled nature of ITA allows both forms of latency to be exploited efficiently. These techniques reduce the overall computation significantly. Of course standard circuit simulators solve every node at every timepoint and all nodes converge simultaneously.

2.7. Implementation in SPLICE

The analysis techniques described in the previous sections have been implemented in SPLICE1.7. The details are described in this section with

special attention given to areas where further optimization would improve the simulator performance.

SPLICE1 has a fixed minimum timestep called the **mrt** (minimum resolvable time). Events can only be scheduled at integer multiples of **mrt**. There is a scheduling threshold parameter called **mindvsch** which is the minimum change in a node voltage over a timestep which causes the fanout elements of the node to be scheduled. The convergence criterion is defined by two parameters called **abstol** (absolute tolerance) and **reltol** (relative tolerance).

2.7.1. Program Flow

The program flow has not changed since the SPLICE1.3 release. The details of the processing may be found in [1, 3] and are not repeated here. The data structures of the ITA as implemented in SPLICE1.7 are given in APPENDIX II. The general program flow for electrical analysis is as follows:

```
set all nodes to their initial values ;
schedule all FOL's at time 0 ;  #FOL = FanOut List of a node
t_n ← 0 ;
while ( t_n < TSTOP ) {
   foreach (FOL in the queue at the current timepoint) {
      foreach (element in the FOL) {
           foreach (output node of an element) {
                process node ;  #see next section for details
                schedule FOL if necessary ;
           }
      }
   }
   plot all requested active nodes ;
   t_n ← t_n + 1 ;
}
```

2.7.2. Details of Node Processing

A subroutine in SPLICE1.7 processes all electrical nodes, calculates the new node voltage, decides whether the node has converged and determines

whether subsequent scheduling is necessary. A high-level pseudo-code description of the routine is as follows:

```
begin
# Iterated timing analysis algorithm in SPLICE1.7
# Node processing sequence
    obtain next node m;
    if (first time processed at new timepoint) {
            use last two points to perform linear prediction ;
            convflg=false ;
    }
    Gnet = Inet = 0 ;
    for ( each fanin element at node m ) {
            compute equivalent conductance Geq;
            compute total current flowing into node Ieq;
            Gnet = Gnet + Geq;
            Inet = Inet + Ieq;
    }
    calculate ΔV ; #change in voltage over an iteration
    V_n^(i+1) = V_n^(i) + ΔV ; #new node voltage
    DV = | V_n^(i+1) − V_(n−1) | ; # change in node voltage over one timestep
    if (ΔV < tolerance) { # node has converged
      if (convflg = false) { #have not converged at this timepoint before
            if ( DV > mindvsched ) { # node change is significant
                    schedule current fol at T_(n+1) (future) ;
                    schedule fol of node at T_n (now) ;
                    convflg = true ;
            }
            else { # node change is not significant over one timestep
                    do nothing ;
            }
      }
      else #have converged previously at this timepoint
            do nothing ; #break any feedback loops
    }
    else { # node has not converged so keep processing
            convflg = false ;
            schedule current fol at T_n (now) ;
            schedule fol of node at T_n (now) ;
    }
# Finished this node for this iteration
return
end
```

2.7.3. Element Models

SPLICE1.7 has built-in models for resistors, linear capacitors (floating and grounded), diodes and MOS transistors. The IIE method is used for

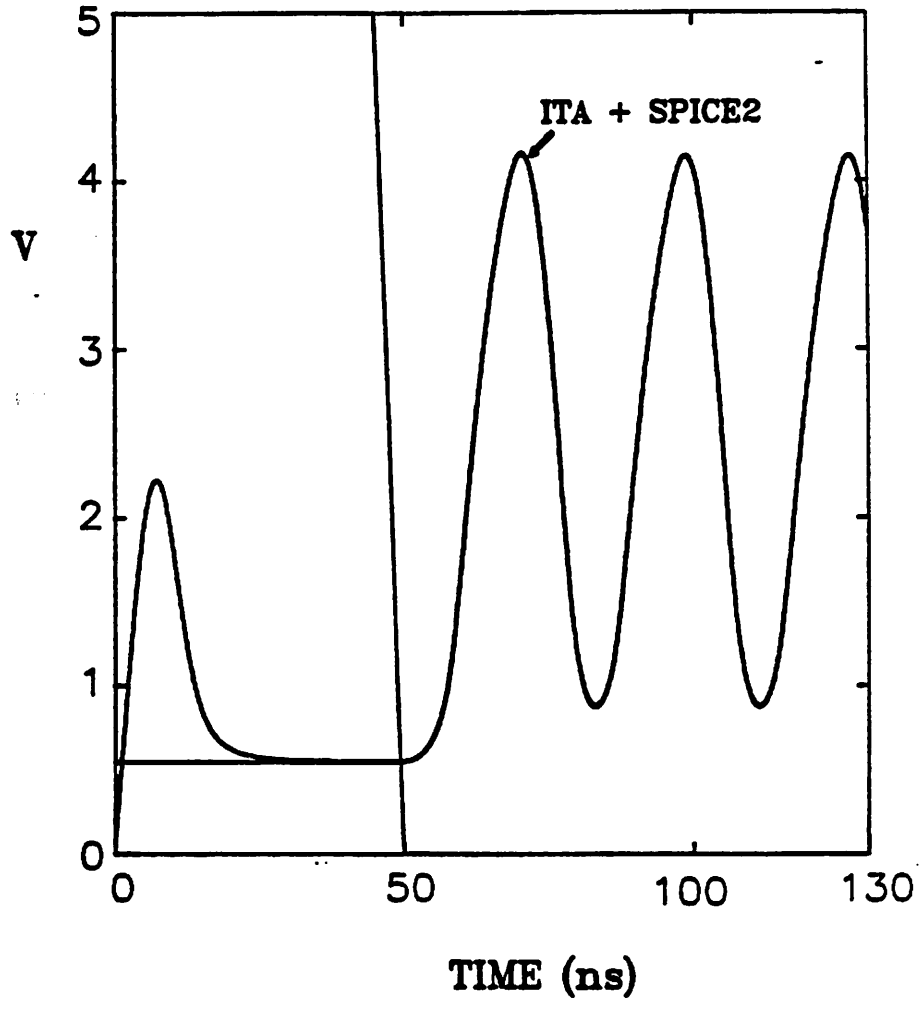
floating capacitors [25] and the first-order Schichman-Hodges model [26] equations are used to model MOS transistors. The model equations for each device are given in APPENDIX III.

Each electrical element has a corresponding program subroutine. The subroutine evaluates the linear equivalent model for each nonlinear device and returns it to the calling routine. As mentioned previously, the intercept current calculation can be avoided by a simple reformulation of the equations. Using this approach, the conductance and the *total* current at a given operating point is returned by each subroutine. The calculation of the equivalent model assumes that all other nodes have ideal *constant* voltage sources attached to them, except in the case of floating capacitors, since IIE is used.

2.8. ITA Simulation Results

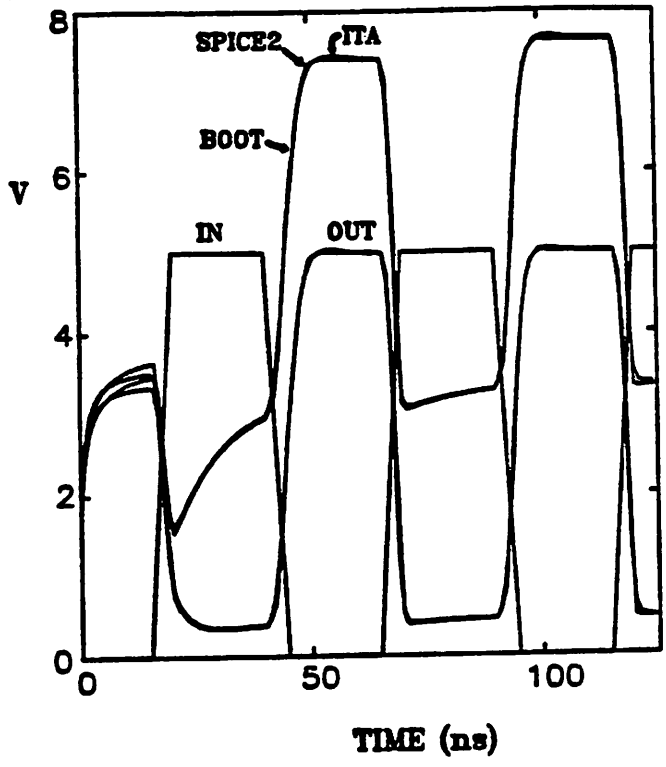
This chapter has been concerned mainly with simulation accuracy and would not be complete without a comparison of ITA with SPICE2. Fig. 2.6 shows the simulation results obtained for the ring oscillator, 2-input NAND and boot-strapped inverter circuits described earlier. As indicated by the results, SPLICE1.7 produces results which are indistinguishable from those obtained by SPICE2 except at timepoints near time zero due to different initial value assumptions. Therefore, circuits which handled inadequately using NTA do not pose a problem to ITA in terms of accuracy.

The run-times of the 3 examples do not demonstrate the speed advantage of ITA because the circuits are all very small with dense G matrices and small circuits tend to be very active. The selective trace feature in SPLICE is a significant advantage in very large circuits.

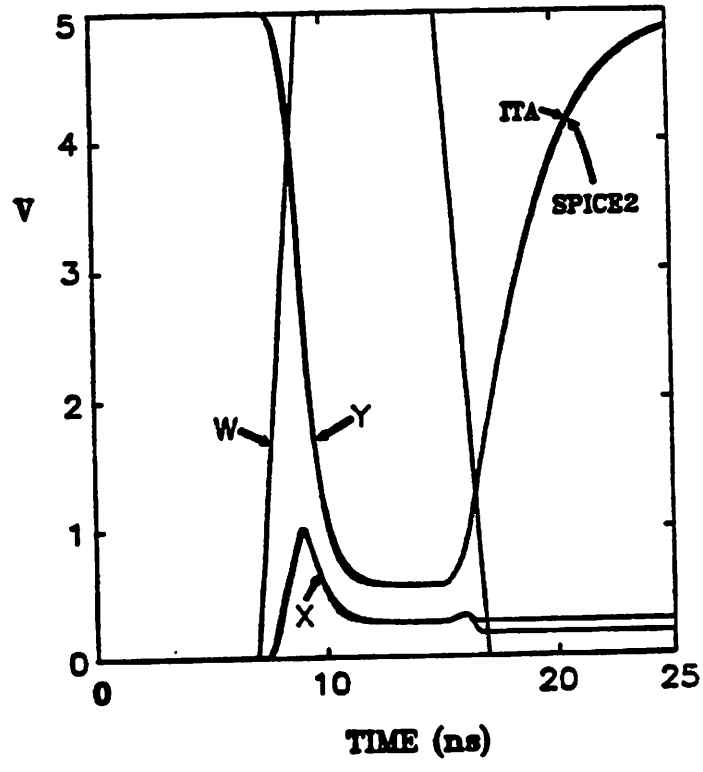


(a) Ring Oscillator Output.
The circuit is shown in Fig. 2.2(a)

Fig. 2.6 : A comparison of the accuracy of ITA vs. SPICE2.
NTA had problems with each circuit.



(b) Boot-strapped Inverter Output.
The circuit is shown in Fig. 2.4(b)



(c) NAND circuit Output.
The circuit is shown in Fig. 2.4(a)

2.9. Optimizations in the Present Implementation

While the data structures used in SPLICE1 are well-suited to handle circuit, timing and logic simulation concurrently, they are not ideal for ITA. If a separate program were written to perform ITA, several optimizations could be made to improve the program performance.

For example, some nodes may be reprocessed after they have converged because there may be several paths to the same node through different elements. A node may also be processed many times in succession before another node is processed (i.e., two or more Newton iterations). Furthermore, there are many levels of indirection which must be traversed in order to reach a node, as shown in a previous section.

These problems can be eliminated by scheduling and processing *nodes* as opposed to fanout lists. One such scheme which uses two buffers, E_A and E_B, avoids reprocessing a node before all other active nodes are processed:

```
put all nodes in event list E_A(0) ;
t_n <-0 ;
while ( t_n <TSTOP ) {
        k ←0 ;
        while ( event list E_A(t_n) is not empty ) {
                foreach ( i in E_A(t_n) ) {
                        obtain ΔV ;
                        v_i^{k+1} = v_i^k + ΔV ;
                        if ( |v_i^{k+1} − v_i^k| ≤ ε ) { i.e., if convergence is achieved
                                add node i to list E_A(t_{n+1}) ;
                        }
                        else {
                                add node i to event list E_A(t_n) ;
                                add fanout nodes of node i to event list E_A(t_n)
                                  if they are not already there ;
                        }
                }
                E_A(t_n) ← E_B(t_n);
                E_B(t_n) ← empty ;
        }
        t_n ← t_{n+1} ;  t_{n+1} = next timepoint
}
```

Another shortcoming of the current implementation is that if a node does not converge at a timepoint, the program simply stops execution. The user must decrease the timestep manually and re-run the entire simulation. An automatic internal timestep control mechanism would be useful not only for the convergence problem but also for error control. If the error is small at a particular timepoint, then the timestep could be increased. If the error is too large, the timestep could be decreased. The nodes would then be re-evaluated at the new timepoint. Hence, the timestep could be computed based on an estimate of the Local Truncation Error. In fact, each node could have its own mrt, independent of other nodes, as long as some consistency is maintained in the simulation between different nodes. Unfortunately, dynamic timestep control requires the ability to "backup" in time (i.e., a buffering of previous results for each node) and requires a modification of the data structures to allow successive refinement of the mrt (minimum resolvable time) in the time queue [27]. For this reason, it would require a considerable amount of effort to test this scheme in the current SPLICE1 environment.

CHAPTER 3

3. Enhancements to the Logic Analysis

3.1. Introduction

The improvements in the logic analysis of SPLICE1 are described in this chapter. The starting point for this work was SPLICE1.3. It had the following features:

- a 4-state logic model (0,1,X,Z)
- fixed assignable rise and fall delays on all gates
- unidirectional and some bidirectional elements handled.

There have been a number of changes in the logic analysis since the SPLICE1.3 release. These changes were made to alleviate some of the problems in the previous version and to facilitate conversions in the mixed-mode environment.

The new version is SPLICE1.7 which features:

- a new MOS-oriented state model
- a fanout dependent delay model
- unidirectional and generalized bidirectional element processing.

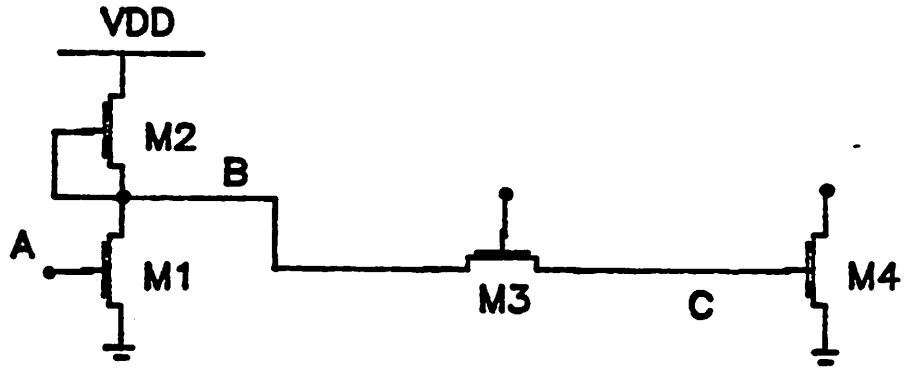
The logic analysis is performed using a relaxation-based method, similar in nature to the electrical analysis. In fact, the logic analysis can be thought of as an implementation of non-iterated timing analysis (see Chap. 2) with simplified element models. Each logic node carries information about the node voltage and the equivalent conductance-to-ground, as does the electrical node. Therefore, the mixed-mode interface is defined in a consistent manner in SPLICE1.7.

This chapter begins with a description and definition of the new state model. Following this, the delay model is described. Next, the "spike" detection and handling procedure is presented. A spike is a pulse at a node of shorter duration than the minimum width necessary to trigger subsequent gates. This is usually an error condition which must be identified and reported to the user. In the next section, the important issues pertaining to the MOS transfer gate are reviewed. The transfer gate (or transmission gate, or pass transistor) is the source of many MOS modeling problems at the logic level and the reason for this will become clear in this chapter. The logic analysis algorithm will then be presented in the section which follows. SPLICE1.7 can also be used to perform switch-level simulation [9, 10] and this is described in the last section. Background material on MOS logic simulation may be found in reference [3].

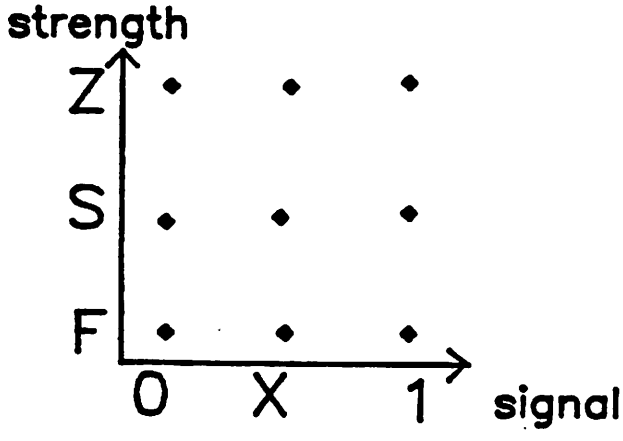
3.2. The State Model

3.2.1. A MOS-oriented Logic Model

Most modern logic simulators handle the problems specific to MOS integrated circuits by including the notion of *signal strength*[7, 8, 9, 10] in the logic model. The rationale for this has been presented in a previous publication [3]. Strength is an abstraction of the large-signal conductance from a node to ground or from a node to a supply voltage. It can be associated with the output of a gate or it can be an attribute of a node. For example, in the inverter of Fig. 3.1 (M1 and M2), the driver transistor with its gate input at $5.0V$ represents a very low resistance path from Node B to ground. In MOS logic model terms, this is referred to as a "forcing 0" or "driving 0". Similarly, the load transistor represents a sizeable resistance from Node B to VDD



(a)



(b)

Fig. 3.1 : The circuit in (a) illustrates the use of the strength-oriented MOS model. The graph in (b) shows the relationship between the strengths and levels in a 9-state logic model.

(approx. 20kΩ to 40kΩ) and this is referred to alternatively as a "soft 1", a "resistive 1" or a "weak 1". If transistor M3 is turned "OFF" (that is, if the gate voltage is zero for an NMOS transistor), Node C goes into a "high-impedance" condition which represents a third distinct strength. The relationship between strengths and levels are shown in Fig. 3.1(b). Although most simulators are based on these three strengths, SPLICE1.7 allows up to 2^{16} -1 strengths for two reasons:

- there is a requirement for more than three strengths when modeling the interaction of several transfer gates with differing W/L ratios, typically found in bus contention situations.
- it provides a mechanism for consistent signal representation in the logic domain for schematic or mixed-mode simulation [22]. If information about the effective conductance to ground is stored with each electrical node, this information could be converted to a strength value and passed on to the logic node, along with the voltage information, whenever there is a requirement to do so. Conversions in the opposite direction can be performed in a similar manner. In this way, simulation accuracy can be maintained in the mixed-mode environment.

3.2.2. State Model Definition

The state model used in SPLICE1.7 is now formally defined :

- A state is composed of a logic level, logic strength pair (L,S).

  i.e.,state=(L,S)=(Level,Strength)

- The logic level can be one of three values: logic zero(0), logic one(1) or logic unknown(X). The "0" level represents the low threshold value or ground. The "1" level represents the high threshold value or VDD. The

"X" level represents an undetermined value which could be "0", "1" or some value in between. The logic level field is extracted from the state using the "lev" function. That is,

$$L = lev(state)$$

- The logic strength is an integer value between 1 and some user-specified upper limit. The upper limit has a maximum allowed value of 65,536. In this report, the subscripts F, W and H will be used to denote the largest, middle and smallest strengths respectively in a given range. The strength field is extracted from the state using the "str" function. That is,

$$S = str(state)$$

- An initial unknown, X_i, must be distinguished from an unknown generated during the analysis, X_g. This is done in SPLICE1.7 by defining the initial unknown as follows :

$$X = lev(initial\_unknown)$$
$$0 = str(initial\_unknown)$$

and the generated unknown as follows:

$$X = lev(generated\_unknown)$$
$$0 \neq str(generated\_unknown)$$

The initial unknown is useful to identify nodes which are not exercised by the input pattern used in a simulation. As a post-processing step, these nodes could be reported to the user.

3.2.3. Using the State Model

In a logic analysis, nodes are scheduled to be processed in the time queue in accordance with the activity in the circuit. When a node is

processed, the fanin list (FIL) is obtained from the node data structure (see Appendix II, parts 1,2). Each gate in the fanin list is a potential "driver" of the node (definition of a fanin) but usually only one gate will gain control of the node and determine its final state. The gate with the largest output strength is declared the "winner" and the node adopts the output state of the winning gate. Node contention occurs when two or more gates attempt to drive the same node to different logic levels with the same driving strength. In this case, the node is assigned an X level and the strength of any one of the contending gates. The processing details are presented in Section 3.6.

3.3. The Delay Model

3.3.1. Factors Affecting Switching Delay

Once a new state is determined, the next task is to calculate the time required to reach the new state. In MOS circuits, the switching time is based on many factors which include:

- the basic gate switching time (unloaded)
- the static output loading due to capacitance of elements connected to the node
- the dynamic output loading through transfer gates which are turned "ON" (that is, transistors with their gates at the logic level "1")
- the number of gate inputs
- the shape (rise and fall times) of input waveforms

No logic simulator attempts to incorporate all of the above factors into the delay calculation. On the other hand, it is essential that a logic simulator

include all the first-order effects in the delay calculation. SPLICE1.7 is capable of modeling the effects due to the first four factors. The fifth factor (input waveform shape) is more difficult to handle at the logic level, although it may be a significant factor in many cases.

3.3.2. Delay Model for Simple Gates

The usual modeling procedure for logic simulation is to generate a set of curves similar to Fig. 3.2 for every primitive element (NANDs, NORs, inverters, etc.) using accurate electrical simulation. In this figure, the delay from the input switching point to the output switching point is plotted as a function of output loading and the number of inputs. A step voltage is assumed as the input of the gate. Although not strictly true, the relationships are usually taken to be linear. The y-intercept of each curve represents the intrinsic unloaded gate delay while the slope of each curve represents the gate drive-capability.

Assuming that the above information is available, the following method can be used to calculate delays for simple gates. The first requirement is that a capacitance value be specified on every input and output pin of every gate as part of the model definition. Then the total gate delay can be represented by four parameters : the intrinsic gate delays (tr, tf) and the gate drive-capabilities (trc, tfc),

where

tr =rise time for unloaded gate (intercept)

tf =fall time for unloaded gate (intercept)

trc =gate drive-capability for rising signals (slope)

tfc =gate drive-capability for falling signals (slope)

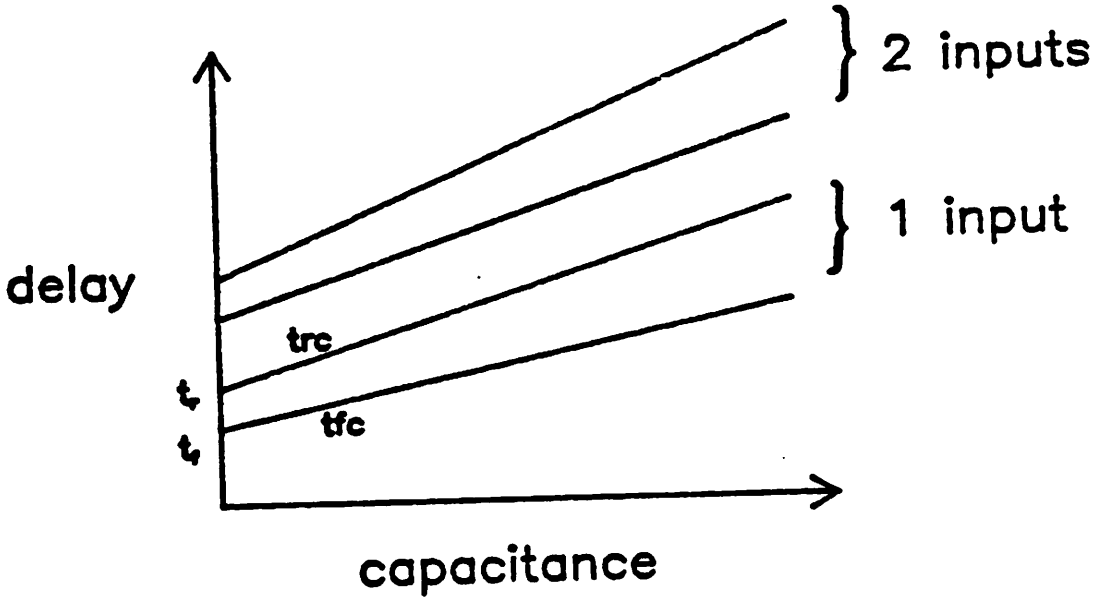


Fig. 3.2 : Typical delay curves generated for a logic gate using electrical analysis

Using these values, the total delay is calculated using the equation:

$$risetime = tr + trc * (node\ \ capacitance) \quad (3\text{-}1a)$$

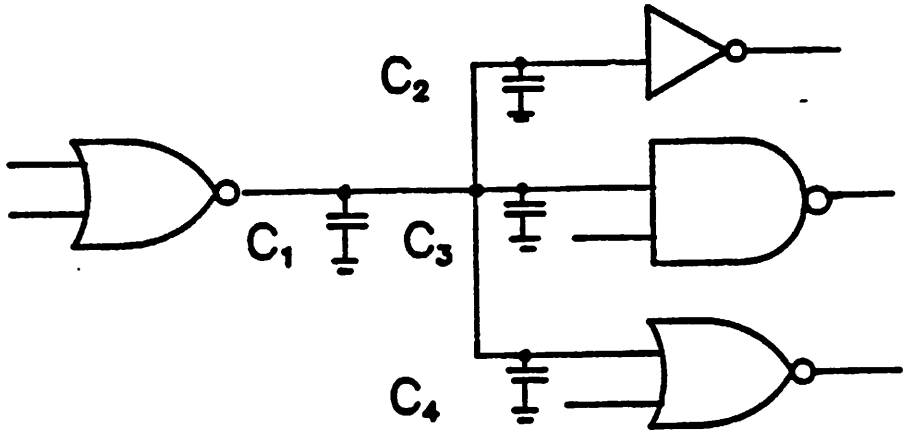
$$falltime = tf + tfc * (node\ \ capacitance) \quad (3\text{-}1b)$$

The node capacitance value is extracted in a pre-processing step by summing the capacitances of all elements connected to a node, and stored with the node data structure (see APPENDIX II, part 1). This process is illustrated in Fig 3.3.

3.3.3. Delay Model for Multi-output Elements

If a multi-output element, such as the flip-flop shown in Fig. 3.4(a), is available as a primitive logic element, each output would have its own set of curves similar to Fig 3.2. The curves for the Q and QB outputs of the flip-flop are shown in Fig. 3.4(b). Then $4N$ parameters would be required to specify the delay, where N is the number of outputs. For the flip-flop there would be 8 such parameters : Qtr, Qtf, QBtr, QBtf, Qtrc, Qtfc, QBtrc, and QBtfc. These parameters would be applied to Eqn. (3-1) to calculate the delay. Using this technique, the delay associated with each output could be handled independently. The overriding assumption is that the rise and fall drive-capabilities (trc,tfc) of the outputs are constant and independent of the inputs.

In certain elements, the delay from a particular input (say, the RESET pin of the flip-flop) to a given output (either Q or QB) is different from another input to output delay (J- or K-input to Q delay). This suggests that, in fact, the intrinsic delay should be a matrix which is indexed by input pin which initiates activity and the output pin being processed. Then the total delay due to loading could be calculated using Eqn. (3-1) and the specific trc and tfc values for each output. This is shown in Table 3.1 below for the flip-



$$C_{tot} = C_1 + C_2 + C_3 + C_4$$

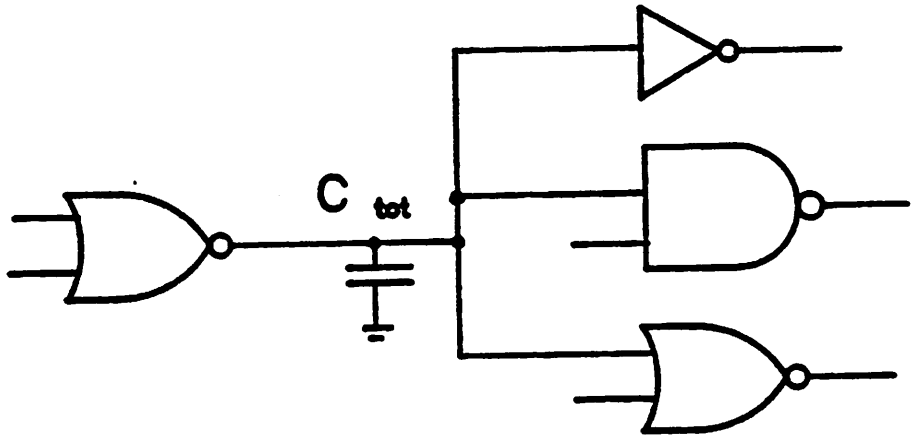


Fig. 3.3 : SPLICE1 preprocesses the input and output capacitances specified for each gate and uses the total capacitance at the node to calculate fanout dependent delays.

flop example.

Table 3.1 Intrinsic Delay Matrix

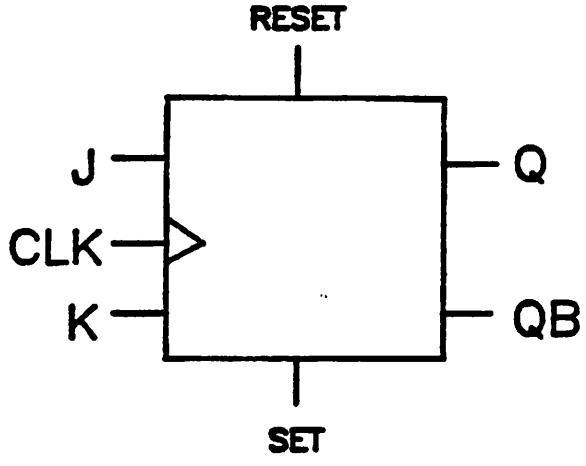
I/O pin	J	K	CLK	RESET	SET
Q	tr=10 tf=10	tr=10 tf=10	tr=10 tf=10	tr=5 tf=6	tr=5 tf=6
QB	tr=10 tf=11	tr=10 tf=11	tr=10 tf=11	tr=5 tf=6	tr=5 tf=6

3.3.4. Delay Models for Transfer Gates

The delay calculation for logic circuits containing transfer gates is more complex than either of the two cases given above. Consider the circuit of Fig. 3.5. The delay from Node A to Node B when the input CLK of the transfer gate makes a transition from "0" to "1" is based on:

- the W/L ratio of the transfer gate
- the drive-capability of gate INV
- charge-sharing between C1 and C2

It is a highly nonlinear situation and therefore difficult to model at the logic level. Charge-sharing cannot be represented properly because of the voltage resolution in the SPLICE1 state model. One method to model this effect is to allow multiple voltage levels in the same way that the impedance levels have been extended. This would facilitate the characterization of charge-sharing but would make the simulator more complicated. The simulator would have to perform transitions from one voltage level to another in a consistent manner. SPLICE1.7 lumps all the nonlinear effects into two values called the turn-on (ton) and turn-off (toff) times. These values do not take capacitive effects into account.



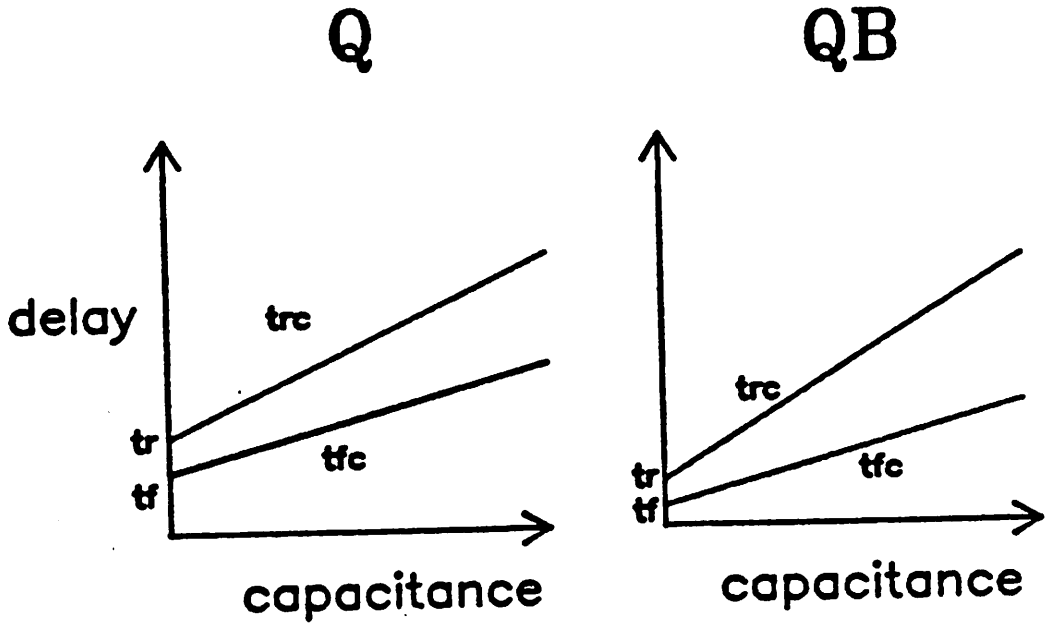


Fig. 3.4 : For the JK Flip-Flop of (a), a set of delay parameters are required for each output, Q and QB, as shown in (b).

Another delay modeling issue concerns transfer gates connected in series as shown in Fig. 3.6. The delay in question is that from Node A to Node E. If all gates are "ON", the circuit can be represented by an RC transmission line. Unfortunately, this is also difficult to model at the logic level. A few alternatives exist to deal with this situation:

- Use a zero delay model through transfer gates when they are "ON" [8]. This is the method used in SPLICE1.7. Unfortunately, the value of delay calculated this way is overly optimistic a Node E.
- Lump capacitances C1, C2, C3, C4 and C5 together and use this value in Eq. (3-1). This is the transition delay for all nodes from the old state to the new one. The value of delay calculated this way is overly pessimistic at Node A.
- Extend the notion of drive-capability of a gate to nodes other than its output node. Since tx1 is "ON", both Node A and Node B are driven by gate INV. Therefore, the delay to A could be calculated as given in eq. (3-1) and the delay to B could be calculated using the equation:

$$risetime = trc_{INV} * (capacitance \ \ at \ \ B) \qquad (3\text{-}2a)$$

$$falltime = tfc_{INV} * (capacitance \ \ at \ \ B) \qquad (3\text{-}2b)$$

To compute the delay to nodes C, D and E, simply apply Eq. (3-2) again using the capacitance at node C, D and E respectively. This approach is better than either of the above methods but is still lacking in accuracy because it does not account for the "ON" resistance of the transistors. One modification which may provide more accuracy is to adjust the values of trc and tfc using the "ON" resistance of the transfer gates and the depth of the node away from the output of the controlling node. This method is promising because VLSI circuits typically contain

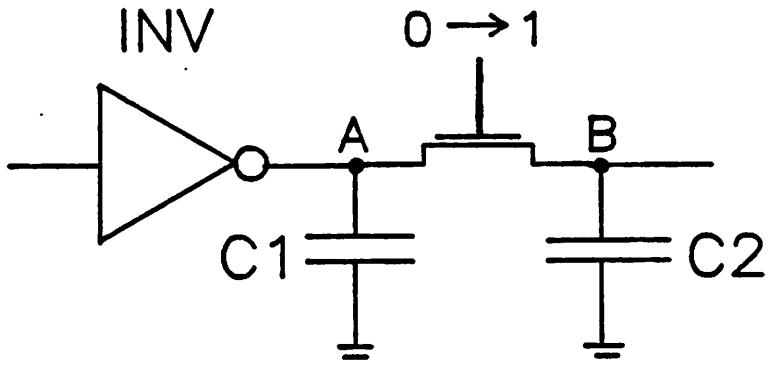


Fig. 3.5 : The delay from Node A to Node B, when the transfer gate turns on, is due to highly nonlinear effects which are difficult to model at the logic level.

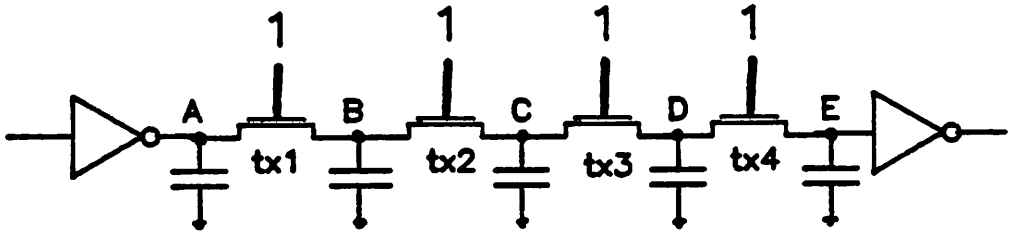


Fig. 3.6 : Delay associated with series-connected MOS transfer gates is difficult to model due to the RC transmission line characteristics.

interconnections which are electrically equivalent to distributed RC transmission lines. This interconnect delay dominates the total delay for very large circuits. It could be modeled the same way as the set of series transfer gates. Therefore, a netlist extractor could provide the simulator with "DELAY" elements,as shown in Fig 3.7, in place of interconnect with delay calculations performed using the modified eq. (3-2) :

$$risetime = TRC * (capacitance \ \ at \ \ node) \qquad (3\text{-}3a)$$

$$falltime = TFC * (capacitance \ \ at \ \ node) \qquad (3\text{-}3b)$$

where

$$TRC = trc * f(resistance, depth)$$

$$TFC = tfc * f(resistance, depth)$$

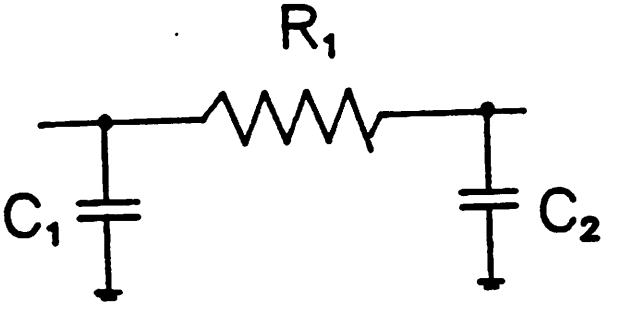


Fig. 3.7 : A proposed equivalent model for a delay element.

3.3.5. Delay to an Unknown Value

The delay calculations in the previous section assume signal transitions from "0" to "1" or "1" to "0". Nodes may, of course, acquire the X level due to contention at the node as described earlier. The question then arises as to when the X level takes effect. The unknown level could be "0", "1" or some intermediate value. Clearly, if the unknown is the previous value, there is no delay. If it is a new logic level there is a rise or fall transition delay. The usual approach is to assume that the unknown value takes affect immediately (as is done in SPLICE1.7) or one time unit in the future.

3.4. Spike Handling

SPLICE1.7 uses an inertial delay algorithm. This means that if a node is scheduled to change at some time in the future T_n, it is held at its old value until that time. Then at T_n, the new value is assigned to the node and the fanouts of the node are processed using this value. A *spike* (commonly referred to as a glitch) occurs if the node is scheduled to change to a different value before it reaches the new value. Spike detection is simple in true-value logic simulation but becomes very complicated when performing fault simulation. When a spike is detected, the event at T_n is dropped, the new event is scheduled at the appropriate time and the user is notified of the glitch. The glitch is not propagated because it usually signifies an error in the circuit design. Therefore, the simulation will continue as if an error did not occur and more meaningful information may be obtained about the correct operation of the circuit. This technique also reduces the amount of work the simulator is required to do since spikes represent activity in the circuit. Therefore, the overall CPU-time will be kept to a minimum by removing glitches from the simulation.

In SPLICE1.7, a fanout list (FOL) can only appear once on the time queue at any given time during the processing. This is a limitation for proper glitch handling, as will be seen in the pseudo-code description of glitch handling which follows. Two different problems are identified which are direct results of the scheduling limitation.

```
#GLITCH HANDLER IN SPLICE1.7
PT = present time
T_next = next time FOL is scheduled
T_last = last time FOL was scheduled to be processed
         (or was actually processed)
if (T_last < PT) { #node was processed in the past
        store new_state ;
        schedule FOL at T_next ;
}
else if (T_last = PT) { #node is scheduled now
        if (T_next ≥ T_last) {
                if (FOL processed) { # PROBLEM : glitch has been propagated
                        update new_state ;
                        schedule FOL at T_next ;
                }
                else { #FOL has not been processed
                #PROBLEM : cannot schedule FOL more than once
                        drop schedule at T_last ;
                        replace new_state ;
                        schedule FOL at T_next ;
                }
        }
}
else if (T_last > PT) { #node is scheduled in the future
        if (T_next < T_last) {
        #reschedule time is earlier than originally scheduled time
                report glitch ;
                drop schedule at T_last ;
                replace new_state ;
                schedule FOL at T_next ;
        }
        else if (T_next = T_last) {
                report glitch ;
                replace new_state ;
        }
        else if (T_next > T_last) { #want to sched in future
                report glitch ;
                drop schedule at T_last ;
                store new_state ;
                schedule FOL at T_next ;
        }
}
```

The problems identified above can be summarized as follows: depending on the order in which nodes are processed at a timepoint, the program may or may not propagate one particular type of glitch, which will be referred to as the Edge glitch or "E" glitch. Therefore, the output of the simulation depends on the order in which the circuit was specified by the user. The "E" glitch is always identified but its propagation is based on node processing order. One way to get around this problem is to use a two-pass approach by first performing a *leveling* operation [28] as a preprocessing step. This simply means that each node should be assigned a value based on its depth from the inputs. Then every node scheduled at a given timepoint should be processed in *ascending* order. This would incur some overhead but would produce the desired results, i.e., the same solution regardless of the order of the input description. At the present time, SPLICE1.7 will identify the glitch and may or may not propagate the glitch depending on the order the nodes are processed.

Another way to eliminate the problem is by modifying the scheduler data structure so that multiple schedules are allowed. Instead of scheduling FOLs, it would be better to schedule structures which point to the FOL. This structure would have to include other information such as the schedule time, and forward and backward pointers to the next and previous schedules of the same FOL in the time queue. This would allow easy access to all the schedules of a single FOL for adding and dropping subsequent events. This proposed data structure is shown in Fig. 3.8. One advantage of multiple scheduling is that the program can be modified to perform parallel fault simulation using this data structure.

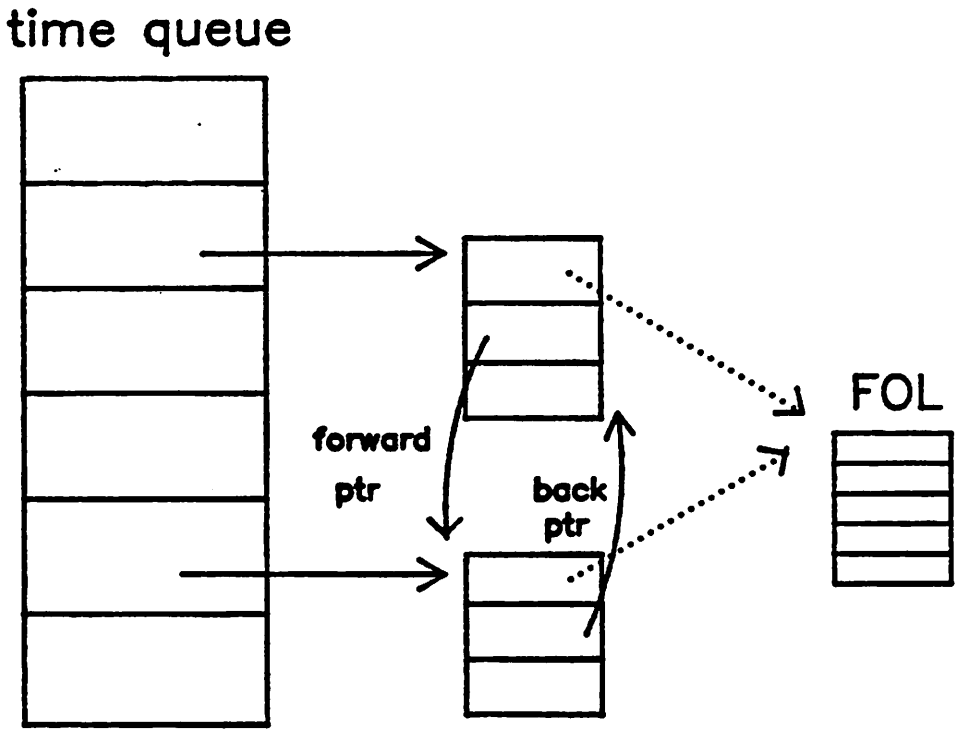


Fig. 3.8 : A proposed data structure to allow multiple FOL scheduling.

A simple circuit which is useful for debugging glitch handling code is the clock generator shown in Fig. 3.9. By adjusting tr and tf for each gate, all possible glitch conditions can be produced. For example, if $tr=tf=10ns$, the "E" glitch can be generated.

3.5. Transfer Gate Modeling Issues

The incorporation of strengths into the state model does not in itself solve all the problems of MOS logic simulation. As described in the previous section, delay modeling is still difficult and the notion of strengths does not provide any leverage in solving the problem. Transfer gates complicate the situation even more because they introduce dynamic loading effects, bidirectional signal flow, node decay, and charge-sharing. In the sections to follow, these and other problems concerning the transfer gate are described and the solutions used in SPLICE1.7 are presented.

3.5.1. Bidirectional Transfer Gates

In general, the transfer gate is a bidirectional element but it is usually found in a unidirectional application. That is, the designer intended signals to flow in one direction through the device. SPLICE1.7 provides unidirectional transfer gates (UTXG) for this purpose, as it simplifies the processing thereby reducing CPU-time.

On the other hand, there are occasions when transfer gates are used in bidirectional applications and therefore the logic simulator must be able to analyze them accurately. There have been a variety of modeling approaches for bidirectional transfer gates (BTXG), including the conventional approach of two unidirectional elements back-to-back as shown in Fig. 3.10. This

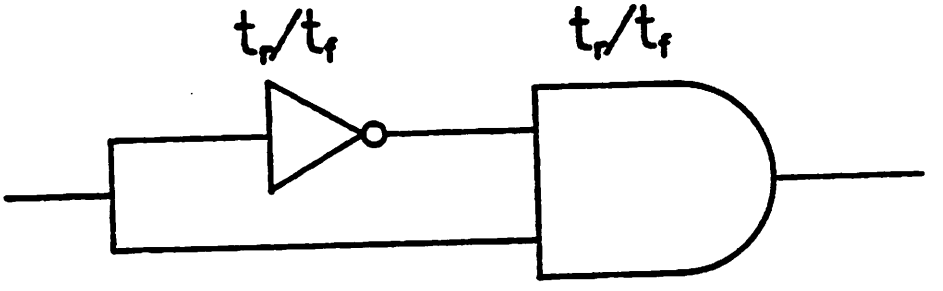


Fig. 3.9 : A simple circuit which can be used to generate the various glitch conditions by adjusting the values of tr and tf.

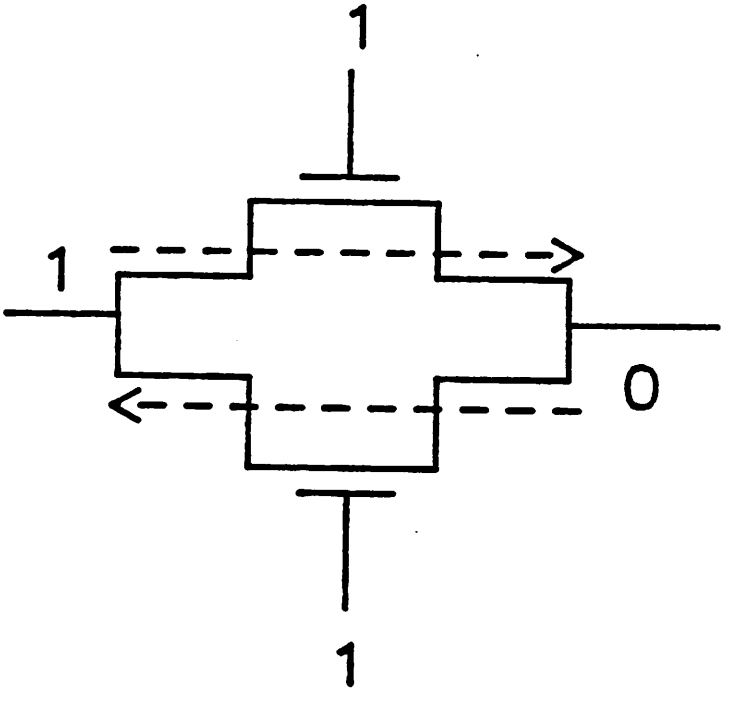


Fig. 3.10 : A bidirectional transfer gate model which employs back-to-back unidirectional transfer gates. The usual problem associated with this approach is that contentions may not be resolved properly.

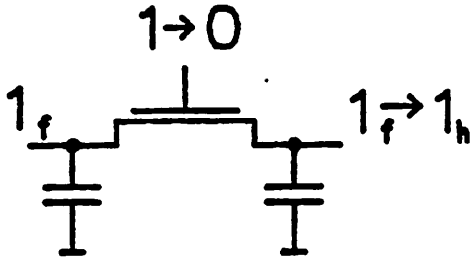
approach can lead to inconsistencies when different logic values are on opposite sides of the element, as is the case in Fig. 3.10. Each value can flow through the BTXG and reach the opposite side and these errors can percolate further through the circuit producing incorrect results. One simple way to process BTXG's in a consistent way is to introduce the concept of composite node relaxation (CNR). In this method, every node connected through transfer gates which are "ON" are considered to be the same node for processing purposes. All fanin lists for the composite node are combined into one list and a new state is determined based on the composite fanin list. Since all nodes connected by "ON" BTXG's are considered the same node, there is no delay between them.

3.5.2. Unknowns at Gate Inputs

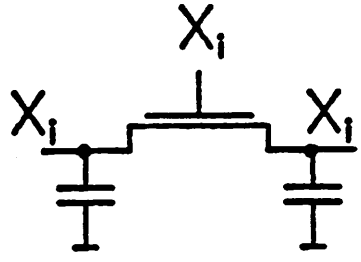
Another problem in modeling transfer gates is due to unknowns at gate inputs. The problem is identified in Fig. 3.11. Normally, if the transfer gate is "ON" and then shuts "OFF", the output Node A retains its previous value but is reduced in strength (goes to the *H* strength). This is shown in Fig. 3.11(a). There are three cases to consider in conjunction with unknowns at transfer gates.

CASE 1 : Fig. 3.11(b) indicates the situation at the beginning of the simulation. Virtually all gate inputs, except for the ones that have been initialized explicitly, are in the initial unknown condition. In this situation, the gate may or may not pass signals.

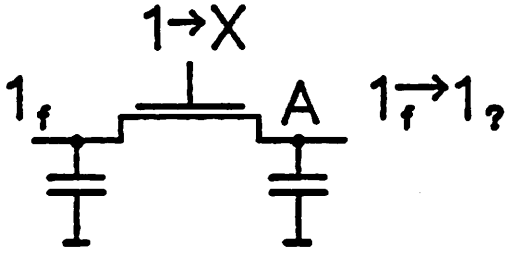
CASE 2 : The second situation occurs when there is a logic "1" at the input and it changes to a logic "X". In this case, the level at the output remains the same but the strength is not known. This is illustrated in Fig. 3.11(c).



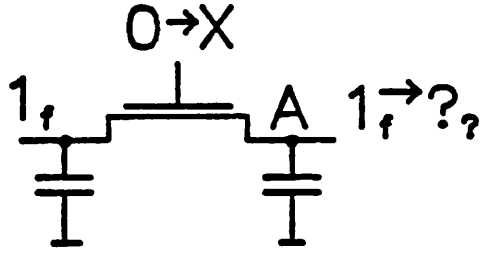
Normal operation with known states at gate input



Case 1 : initial unknown produces unknown levels and strengths



Case 2 : 1 -> X produces an unknown strength at the output, although the level is known



Case 3 : 0 -> X produces an unknown level and strength

Fig. 3.11 : Various cases when processing transfer gates with X at the gate input

CASE 3 : The third situation is the reverse of the second. Here, the input goes from logic "0" to logic "X". Both the value and strength may change. This is shown in Fig. 3.11(d).

There are a few alternative methods to handle unknowns at gate inputs.

(1) a pessimistic approach is to generate X_f at the output so that it will be propagated further. This may produce incorrect circuit operation if CASE 2 is considered, but is the easiest to implement.

(2) another approach is to have the notion of unknown strengths. Using this model, CASE 2 could be handled by setting the output node to its previous value with an unknown strength. This introduces some complications in the way the simulator processes nodes. Some bit pattern would have to be selected for unknown strengths. It is not clear how this special strength value would interact with other strengths.

Currently, method (1) is used in SPLICE1.7 and other methods are under investigation [29].

3.5.3. Node Decay

When a node acquires the H strength, it retains the previous state on the capacitance at the node. In physical terms, charge is trapped at the node but there are parasitic resistive paths from the node to ground or VDD. Therefore, the node will eventually lose its value and it will become unknown. This is referred to as *node decay*. The time constant for the decay is large but finite.

It is useful to include node decay as part of the simulation, especially for dynamic circuits. One way to do this is to detect the H strength at a node and schedule the node to decay after a specified amount of time by setting a

special flag at the node. If the node is not redriven before this time, the node is placed in the unknown state. If the node is redriven, the scheduled event would be dropped and processing would continue in the normal way. Unfortunately, the program would incur an excessive amount of scheduling and de-scheduling overhead, especially in the case of dynamic MOS circuits. Moreover, in SPLICE1.7, most of the scheduled nodes would be put into the pool (see Appendix II, part 5). The pool is an overflow area designed to store all schedules which are greater than 200 timepoints in the future. If node decay is processed as suggested above, the scheduled decay events would all be placed in the pool and eventually overflow the limit of the pool area. Clearly this is not a suitable approach.

An alternative approach, proposed by Boyle [30], is to simply store the decay time along with the node data structure and avoid scheduling altogether. Anytime the node is redriven, this value could be compared to the current time. If the current time is greater than the decay time, a warning message could be placed in a file, if the user has requested decay errors. Then processing would continue as if node decay had not occurred. It is not useful to simulate the circuit under decay conditions because it is usually a design error. Therefore, it is simply flagged as an error and then ignored for the remainder of the simulation.

3.6. Logic Simulation Implementation Details

3.6.1. General Program Flow

The following is a high-level pseudo-code description of the general program flow during a logic analysis. Note the parallel between the ITA program

flow described in the previous chapter and the code below:

```
set all nodes to their initial values ;
schedule all FOL's at time 0 ;  # FOL = fanout list
t_n=0 ;
while ( t_n≤TSTOP ) {
   for (each FOL in the queue at the current timepoint) {
      for (each element in the FOL ) {
           for (each output node of an element) {
                process node ;  #see next section for details
                schedule FOL if necessary ;
        }
      }
   }
   plot all active nodes ;
   t_n←t_n+1 ;
}
```

3.6.2. Node Processing Details

```
# LOGIC NODE PROCESSING DETAILS
#
begin
  current_state ← (X,0) ;
  place node in CNL ; # CNL = composite node list
  for (each node in the CNL) {
     for (each element in the FIL) { # FIL = fanin list
          if (element = BTXG) & (gate = "ON") {
              place node in CNL ;
          }
          else {
            determine output_state (L,S) of element ;
            intend_state ← output_state ;
          }
          if ( str(intended_state) > str(current_state) )
            current_state ← intended_state ;
          else if ( str(intended_state) = str(current_state) )
            if ( lev(intended_state) ≠ lev(current_state) )
              lev(current_state)  ←  X ;
     }
  }
  new_state  ←  current_state
  if (new_state ≠ old_state) {
    for (each node in CNL) {
         calculate delay(old_state,new_state) ;
         call GLITCH HANDLER to schedule FOL ; # FOL = fanout list
    }
  }
end
```

3.7. Switch-level Simulation

The definition of UTXG and BTXG elements (given in 3.5.1) allows switch-level simulation [9, 10] to be performed using SPLICE1.7. Loads can be modeled using a UTXG with a "weak" output strength. Drivers can be modeled using a UTXG with a "forcing" output strength. Either a BTXG or a UTXG can be used for pass transistors depending on the application. Other floating transistors must be BTXG's. If "ton" and "toff" are specified as 1 unit of time, then a unit-delay switch-level simulation will be performed by SPLICE1.7. For obvious reasons, delays at the switch-level cannot be modeled in the same way as it is currently done is SPLICE1.7 for standard Boolean gates. Two approaches have been proposed to introduce detailed timing information at the switch-level using a resistive simulation model [31, 32]. These methods are under investigation at the present time.

CHAPTER 4

4. Examples and Results

In this chapter, a number of simulation results and program performance statistics of SPLICE1.7 are presented. Five aspects of the program are examined in the sections to follow. These are :

- the program performance statistics such as processing speed for the electrical and logic analyses, typical storage requirements per element, iteration counts, etc.
- the identification of bottlenecks using profilers
- the factors which affect the run-times such as **mindvsch**, **sor**, **mrt** and floating capacitors
- SPLICE1/SPICE2 comparisons for execution speed, memory requirements and simulation accuracy
- mixed switch, logic and electrical-level simulations

The simulations were carried out on the following circuits:

(1) **Digital Filter Circuit** : This circuit was obtained from [1]. It is the control logic for a digital filter circuit. There are 705 MOS transistors and 393 nodes in the circuit. The simulation period is 4μs.

(2) **Counter-Decoder-Encoder Circuit** : This circuit is a combination of a 4-bit counter driving a 4:16 decoder and a 16:4 encoder. It is referred to as the CDE circuit in the rest of the chapter. The switching characteristics were based on the specifications provided in a TTL Handbook [33]. The circuit has 1,326 MOS transistors and 553 nodes and is the largest

circuit simulated so far. The simulation period is also 4μs [34].

(3) **NMOS Operational Amplifier** : This circuit was obtained from [35]. It was designed as part of a phase-locked loop circuit. This circuit is used to illustrate the capability of ITA when simulating analog circuits.

(4) **Boot-strapped Inverter Circuit** : This circuit was described earlier in Chap. 2. It is illustrated in Fig. 2.4(b). The circuit is used to examine the effects of a floating capacitor element in an ITA simulation.

(5) **Industrial Microprocessor Control Circuit** : This is the critical path through the control circuitry of a μP designed using NMOS technology.

(6) **Industrial 64K Ram Circuit** : This is a portion of a high-speed 64K static RAM circuit designed using CMOS technology.

(7) **4 x 5 Multiplier Circuit** : This circuit uses standard multiplier structure. It features novel exclusive-OR and ADDER functions designed by Kuninobo [36]. This example is used to illustrate mixed-mode simulation and to compare the run-times associated with transistor-based simulation at the electrical and switch levels.

4.1. Program Performance Statistics

In order to predict the run-times and memory requirements of the program SPLICE1.7, the program execution speed and memory usage statistics are required. These statistics have been tabulated below for both the electrical and logic simulators.

Electrical Simulation Statistics

Node Evaluations	400 nodes/sec.
SOR-Newton Iterations (no floating caps)	3-5 iterations/node
SOR-Newton Iterations (with floating caps)	6-20 iterations/node

Electrical Element Storage Requirements

Elements	Type	Words Required
Transistors	Load	3 x no. of loads
	Driver	4 x no. of drivers
	Transistor	5 x no. of transistors
Capacitors	Grounded	0
	Floating	3 x no. of capacitors
Resistors		3 x no. of resistors

Element Model	Type	Words Required
Transistors	Load	13 x no. of loads
	Driver	13 x no. of drivers
	Transistor	14 x no. of transistors
Capacitors	Grounded	1 x no. of grounded capacitors
	Floating	3 x no. of floating capacitors
Resistors		3 x no. of resistors

Logic Simulation Statistics

Node Evaluations	650 nodes/sec.

Logic Element Storage Requirements

Elements	Words Required
inverter	3 x no. of inverters
buffer	3 x no. of buffers
AND	~ 5 x no. of ANDs
OR	~ 5 x no. of ORs
NAND	~ 5 x no. of NANDs
NOR	~ 5 x no. of NORs
XOR	~ 5 x no. of XORs
XNOR	~ 5 x no. of XNORs
transfer gates	~ 4 x no. of devices

Model Type	Words Required

inverter	11 x no. of different inverter models
buffer	11 x no. of different inverter models
AND	~ 11 x no. of different AND models
OR	~ 11 x no. of different OR models
NAND	~ 11 x no. of different NAND models
NOR	~ 11 x no. of different NOR models
XOR	~ 11 x no. of different XOR models
XNOR	~ 11 x no. of different XNOR models
transfer gates	~ 8 x no. of device models

Node Storage Requirements

N = number of circuit nodes

Data	Logic Node	Electrical Node
Node list	1	1
Node pointers	N	N
Node data	8N	9N
Node FIL	N	3N
Node FOL	3N	3N
Capacitor	N	N

4.2. Profile Statistics

The SPLICE1.7 program execution times can be reduced somewhat by applying the techniques suggested in Chap. 2. These were based on intuitive arguments. It is important to identify bottlenecks in the program and identify where it is spending most of its time in a quantitative way. A profiler is a modern programming tool which is very useful for this task. It monitors the program during execution and provides information relating to the percentage of time spent in each subroutine. Using this information, the program can be modified in sections where it will provide the most benefit.

The following profile statistics were obtained from a simulation of the digital filter circuit using electrical analysis.

total time: 3196 seconds

time(%)	time(sec.)	no. of calls	name	subroutine task
28.7	916.33	1222883	prtim	processing a timing node
17.3	554.47	2449814	tntxg	evaluate transistor model
14.2	455.32	5799795	getexv	get a value from another node
8.2	231.63	951895	sqrt	perform a square root operation
6.3	203.63	900167	tndri	evaluate driver model
5.1	164.07	658140	tnloa	evaluate load model
4.7	152.62	792936	prelm	process an element in the FOL
1.6	51.92	4001	prfot	process a FOL in the time queue
1.3	40.23	276985	adsfo	add a FOL to the time queue
1.2	37.53	24046	dropf	drop a FOL from the time queue
0.3	9.70	20547	prout	print out a node

It is clear that most of the time is spent processing nodes and evaluating transistor models for the SOR-Newton iteration. Therefore, any speed-up techniques should be applied to these areas of the program. It is expected that, as the program is developed further, information provided by the profiler can be used to significantly reduce execution time, particularly for electrical simulation. Other methods, currently available to the user to reduce the total run-time, are described in the next section.

4.3. Factors Affecting Execution Time in Electrical Simulation

4.3.1. CPU-time vs. MRT

SPLICE1.7 has a user-specified fixed minimum timestep called the **mrt** (Minimum Resolvable Time). The symbol h will be used to refer to the timestep associated with a particular node. In SPLICE1, h is some integer multiple of **mrt**. Although the minimum timestep is fixed, the value of h for a specific node is dependent on the activity at that node. For example, when the node is active, h is equal to **mrt**. Otherwise, h is defined by the time difference between two events at the node. In a sense, the timestep at a node is determined implicitly by the activity in the circuit.

Since there is no explicit timestep control mechanism in SPLICE1, the CPU-time required to perform electrical simulation is a strong function of **mrt**. Fig. 4.1 illustrates this relationship for the CDE circuit. There is an optimum value of **mrt** for this particular circuit at 1ns. At values of **mrt** greater than the optimum, the program iterates longer to produce solution at a particular timepoint. This is due to the fact that a linear prediction generates a poor guess as the timestep is increased and the diagonal terms of the conductance matrix, $\frac{C}{h}$, are reduced thereby weakening the diagonal dominance property of the matrix. In fact, at a very large value of **mrt** the program may not converge at all.

At **mrt** values less than the optimum, the program is forced to do more work during the active periods than is really necessary, based on the time constants in the circuit. Therefore, there is a rapid rise in the curve in Fig. 4.1 below the optimum. It has been observed that at very small timesteps, the curve begins to level off. This is probably due to the fact that the prediction step is very accurate and only one or two iterations are required for convergence.

The relation between CPU-time and **mrt** suggests that, for a given technology, the optimum value should be obtained through experiment and used in all further simulations. Of course, if an explicit dynamic timestep control mechanism is implemented, this would not be necessary.

4.3.2. CPU-time vs. MINDVSCH

In SPLICE1, events at a node are propagated to its fanouts if the change in the node voltage is considered to be significant. If the change is not significant, the fanouts are not scheduled and the node is returned to its

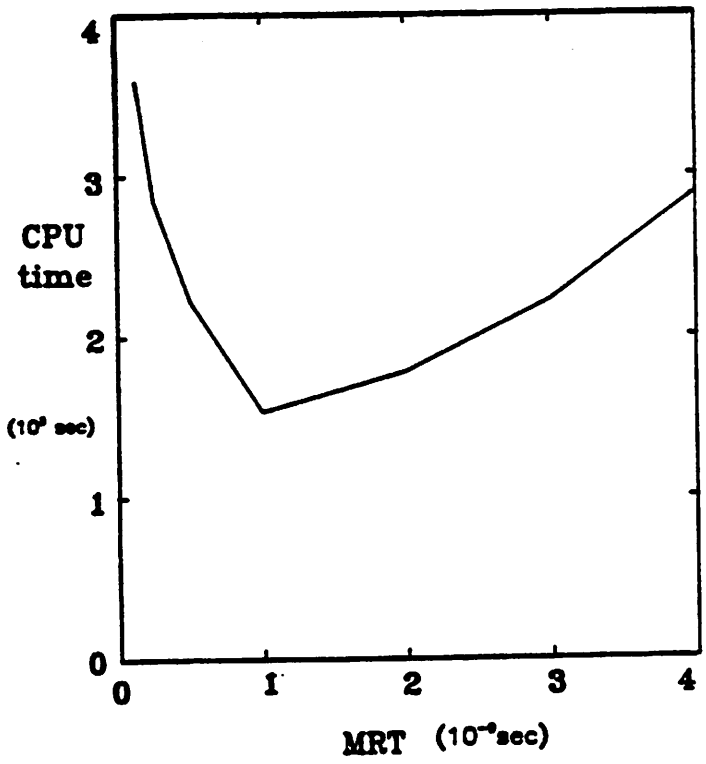


Fig. 4.1 : CPU-time vs. MRT

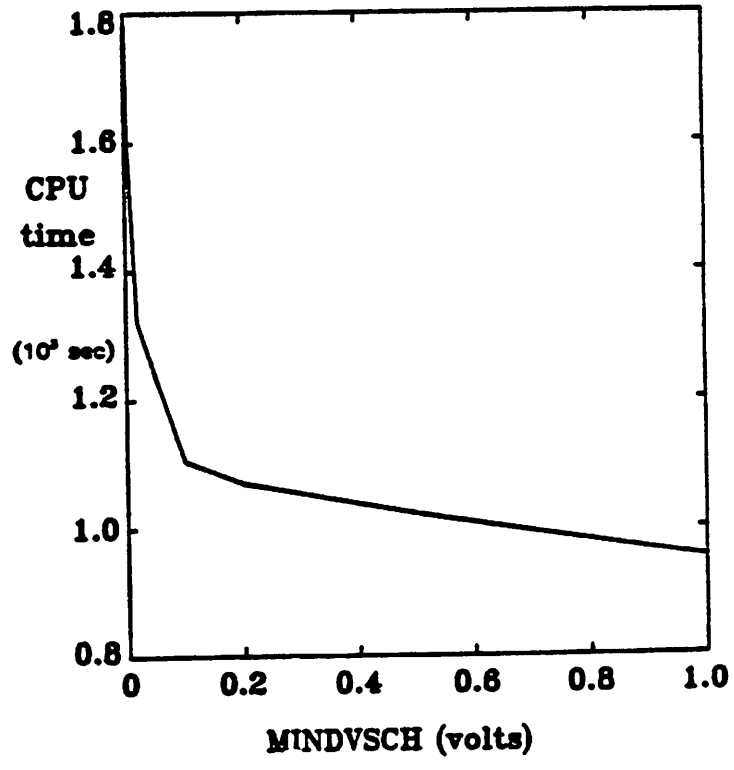


Fig. 4.2 : CPU-time vs. MINDVSCH

original value. This constitutes the event-driven selective-trace feature in SPLICE1. The scheduling threshold parameter, used to determine whether or not the change is significant, is called **mindvsch**. It is specified in units of volts, by the user, for an entire simulation and is the same for every node in the circuit.

The value chosen for **mindvsch** has a profound effect of the simulation results. Careful consideration must be given to select an appropriate value for this parameter. It can be thought of as the minimum voltage change which can affect the fanout elements of a node. Based on experience with the program, an appropriate value for most digital circuits is $1mV$. It is much smaller for analog circuits, particularly if there are high-gain stages.

The effect of **mindvsch** on CPU-time is quite dramatic as shown in Fig. 4.2. As **mindvsch** is increased, the CPU-time goes down. This suggests that the value should be made as large as possible. Unfortunately, some significant events may be accidently dropped if the **mindvsch** is too large resulting in a loss of accuracy. Also, node voltages can only reach a value which is within **mindvsch** of their final value because the remaining voltage change is not considered significant. Hence, if **mindvsch** is too large, there will be errors at the end of each transition.

4.3.3. Effect of Floating Capacitors

As indicated in Chap. 2, floating capacitors no longer pose a problem to the electrical analysis in terms of accuracy but tend to degrade the simulator performance. The factor which determines the amount of degradation is the ratio of the floating capacitor to the grounded capacitor. In order to illustrate the relationship between CPU-time and capacitance ratio, the boot-strapped inverter of figure 2.4(b) was simulated with different values of

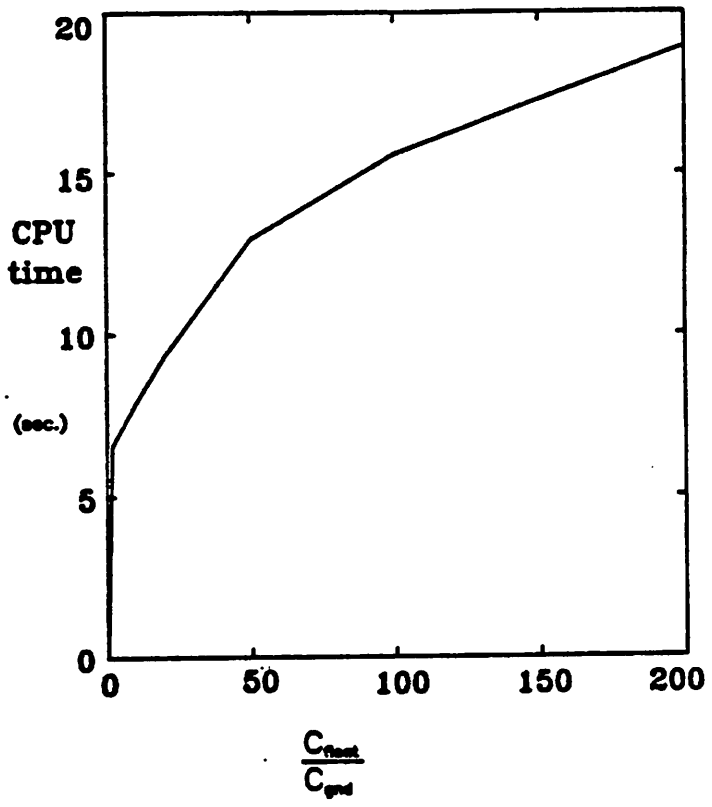


Fig. 4.3 : CPU-time vs. Capacitance Ratio

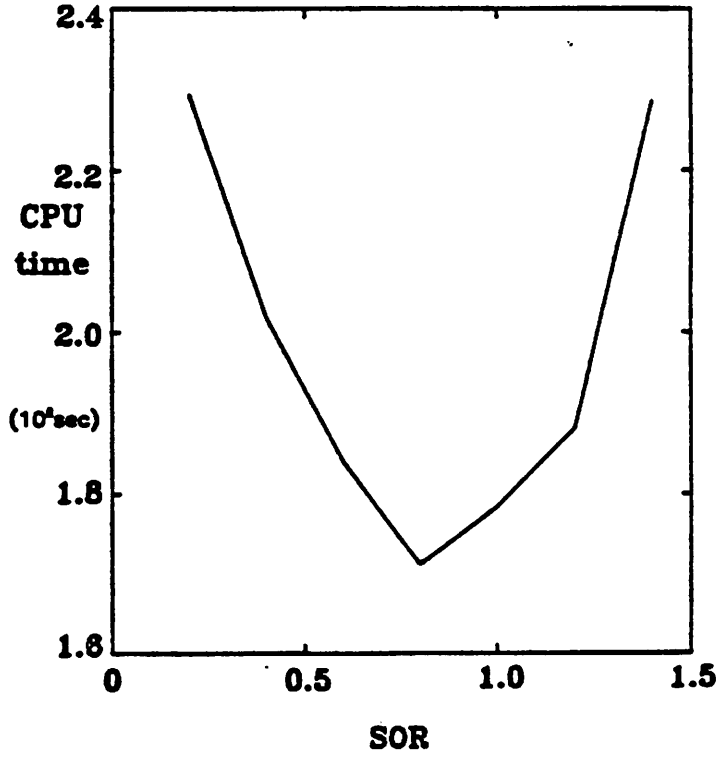


Fig. 4.4 : CPU-time vs. SOR

$\frac{C_{float}}{C_{gnd}}$. The results have been plotted in Fig. 4.3. It is clear from this graph that the relationship obeys a square-root law.

Although the graph indicates that solutions may be obtained regardless of the ratio, this is not true in general. Therefore, if the ratio is too large, the iteration may not converge. Special techniques must be used to reduce the number of iterations required to solve nodes with floating capacitors since this is usually the case. Research is underway to find ways to accomplish this.

4.3.4. CPU-time vs. SOR

The **sor** parameter was introduced in Chap.2 as an acceleration parameter for the nonlinear Gauss-Seidel iteration. This parameter has a significant effect on the CPU-time but does not affect simulation accuracy. Fig. 4.4 shows the relationship between CPU-time and **sor** obtained from simulations performed on the digital filter circuit. There is an optimum value of **sor** which minimizes the run-time of the simulation. In this case, the optimum value is 0.8.

Although the optimum value changes from technology to technology, it is worthwhile to obtain the value experimentally as it may provide a substantial improvement over the standard Gauss-Seidel iteration.

4.4. SPICE2 vs. SPLICE1.7

Five circuits were simulated at the electrical level using SPICE2 and SPLICE1.7 to compare run-times, memory requirements and accuracy. These simulations were performed on a VAX-11/780 under the UNIX operating system. In both simulators, default parameter values were used for the

convergence criteria, integration method, and accuracy tolerance. The values are available in the user guide for each program.

4.4.1. CDE Circuit

A block diagram of the CDE circuit is shown in Fig. 4.5. The details of the blocks are given in [33]. This circuit is large and highly unidirectional in nature. A 4-bit counter provides a sequence of inputs to the 4:16 decoder, the output of which is encoded to 4-bits. The simulation period was 4μs with an **mrt** of 1ns. As shown in Fig. 4.6, the output waveforms have long latent periods. For this circuit, SPLICE1.7 was 66 times faster than SPICE2 and its memory usage was 35 times smaller, for comparable accuracy. More importantly, the SPICE2 simulation required 32 hours whereas the SPLICE1 simulation required only 40 minutes! This represents a *substantial* improvement in speed and allows a simulation of this magnitude to be quite feasible. A small fraction of this speed advantage can be attributed to the fact that SPLICE1 has been tailored for transient analysis, but the the key reason for the improvement is the efficient exploitation of latency.

The simulation results are summarized in Table 4.1. Also included in this table are the simulation results obtained using SPLICE1.3, an earlier version of SPLICE1 which used NTA. It is interesting to note that SPLICE1.7 required only twice as much time as SPLICE1.3 to produce a solution even though it iterates to convergence. SPLICE1.3 uses only one SOR-Newton iteration at each timepoint, but it can reject the new solution if the change in voltage over an **mrt** is considered to be too large, as determined by a parameter called "maxdvstep". This is done to maintain simulation accuracy. For example, if the voltage change between times t_{n-1} and t_n is con-

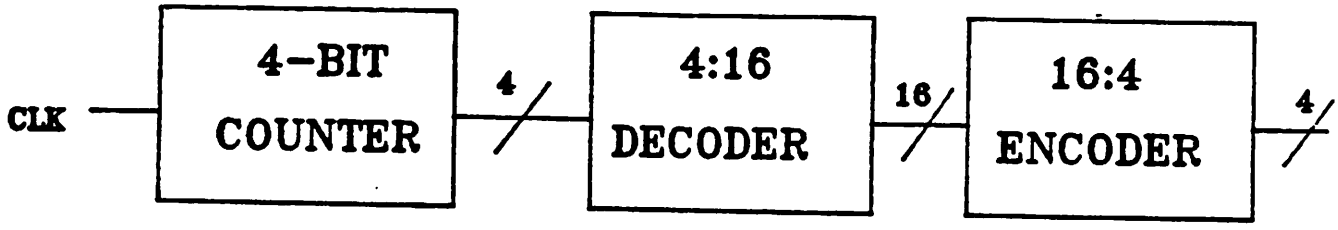


Fig. 4.5 : Block Diagram of the CDE circuit

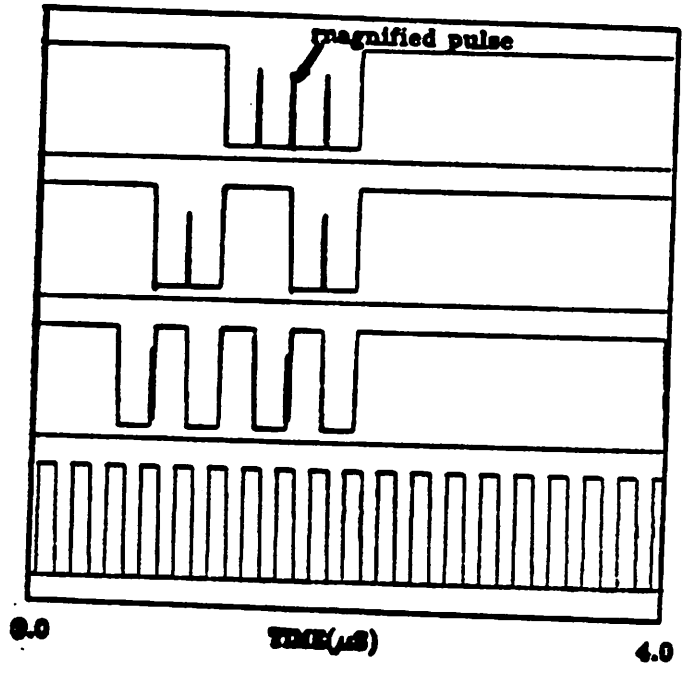


Fig. 4.6 : Output waveforms of CDE circuits. The waveforms feature long latent periods which is an advantage for ITA.

sidered to be too large, the program would cut the timestep by a factor of 4 and perform another series of single iterations at timepoints $t_{n-1}+\frac{h}{4}$, $t_{n-1}+\frac{h}{2}$, $t_{n-1}+\frac{3h}{4}$ and t_n. Further timestep cutting would be done if the voltage change was deemed to be too large over any subinterval. Therefore, many evaluations may be performed to produce the solution a timepoint, although a single iteration is always used at any given point in time. The attempt to maintain accuracy in this manner increases the overall simulation time. Furthermore, on the first iteration at a timepoint, SPLICE1.7 uses previous history to predict a new voltage at a node, thereby reducing the total number of iterations required to converge to a solution.

Circuit Mosfets Nodes	CDE 1,326 553	
	Time (s)	Memory (Kbyte)
SPICE2G	115,840	2,420
SPLICE1.7	1,740	68.9
Ratios	66	35
SPLICE1.3	843	68.9

Table 4.1

Comparison of conventional circuit simulation and ITA for the CDE circuit.

One of the pulses in Fig. 4.6 has been magnified in Fig. 4.7 to compare the accuracy of the three approaches used to simulate the CDE circuit, as indicated in Table 4.1. The pulses from SPLICE1.7 and SPICE2 are centered at 1627ns and 1628ns respectively. This difference would be indistinguishable at the level shown in Fig. 4.6 and it is not clear which is the more accurate solution. In this case, the true solution lies between the SPLICE1.7 and

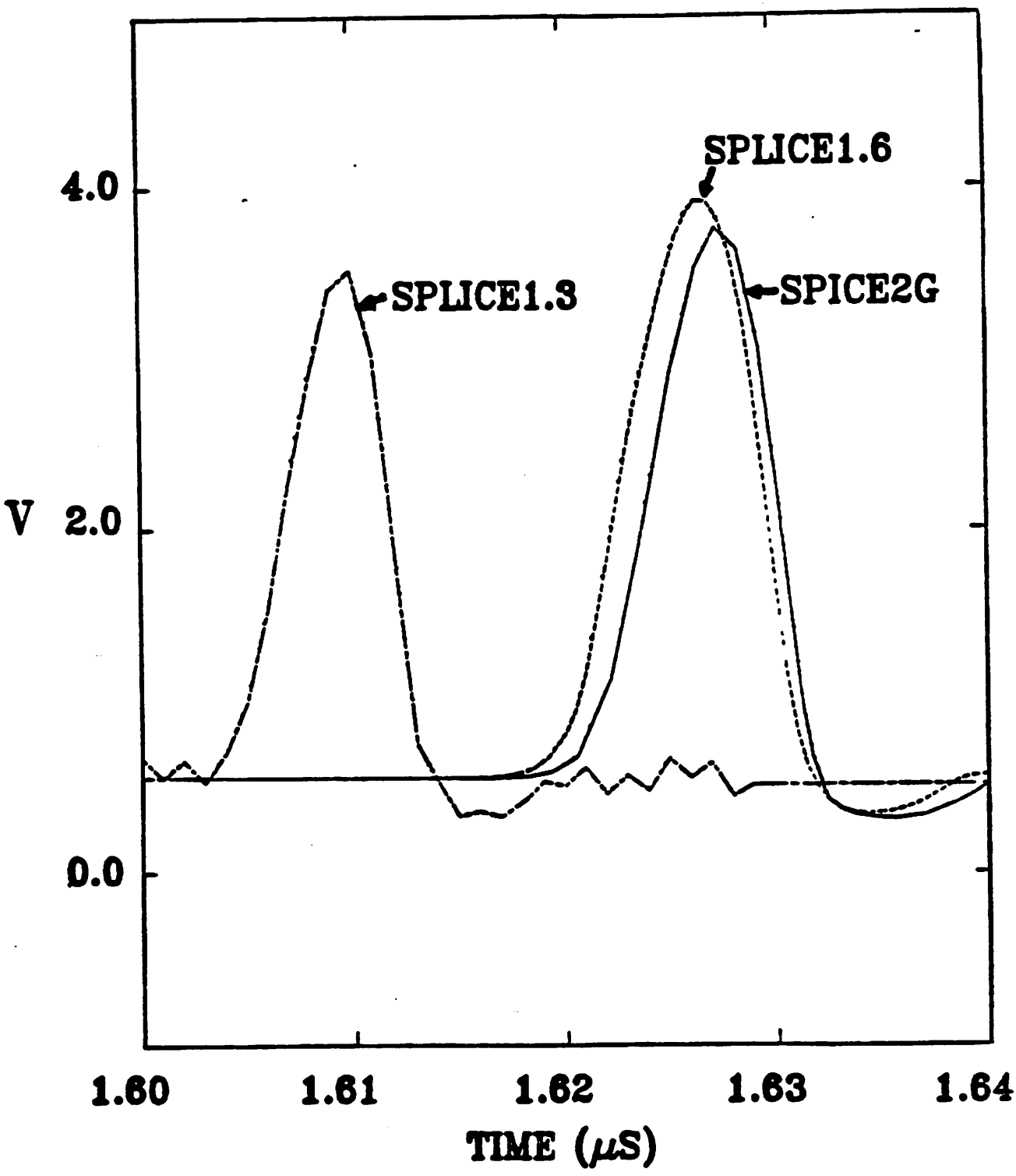


Fig. 4.7 : The pulse shown in 4.6 has been magnified here to compare ITA with NTA and SPICE2.

SPICE2 results. The output of SPLICE1.3 features some numerical noise in the vicinity of 0.5 volts which can be attributed to the single SOR-Newton iteration used in NTA. The pulse generated by SPLICE1.3 is approximately correct in its size and shape but is centered incorrectly at 1608ns, an error of 20ns.

4.4.2. Digital Filter Circuit

The block diagram of the digital filter is given in Fig. 4.8. Further details may be found in [1]. The circuit was simulated using SPLICE1.7 and required 1783 seconds. The original SPLICE1 program, as described in [1], required 453 seconds which is 4 times faster. Clearly, the cost of ITA vs. NTA depends on the size and nature of the circuit, but the key point is guaranteed accuracy in the solution produced by ITA. For this circuit, SPLICE1.7 was 17 times faster and its memory usage was 21 times smaller than SPICE2. This is due to the fact that this circuit is somewhat smaller than the CDE circuit and has much more activity.

Circuit Mosfets Nodes	Digital Filter 705 393	
	Time (s)	Memory (Kbyte)
SPICE2G	30,582	1,038
SPLICE1.7	1,783	48.4
Ratios	17.1	21.4

Table 4.2

Comparison of conventional circuit simulation,

and ITA for the Digital Filter Circuit

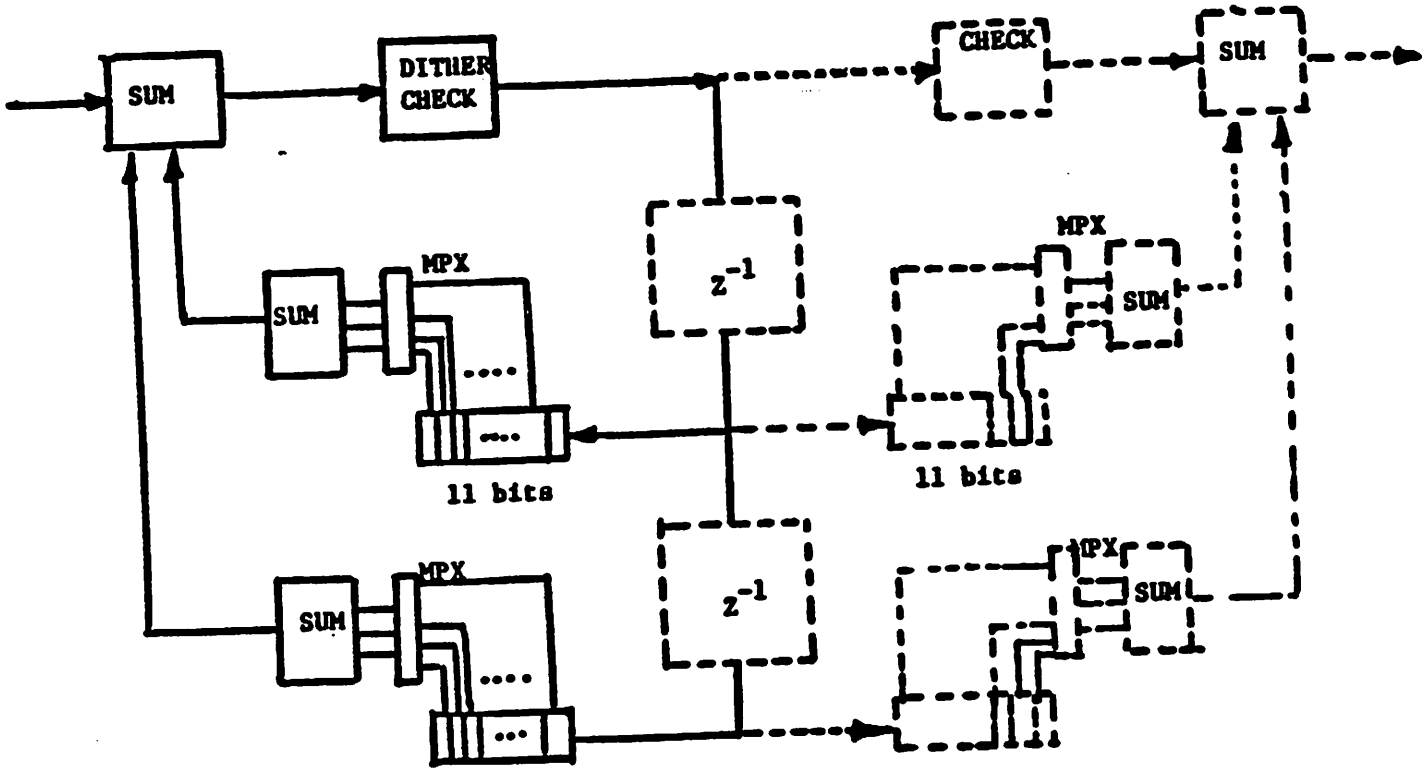


Fig. 4.8 : Block Diagram of Digital Filter Circuit

4.4.3. Industrial uP Control Circuit

This schematic for this circuit is shown in Fig. 4.9. It contained over 100 transistors and 100 diodes and is representative of a typical simulation performed using the SPICE2 program. Although no extra elements were added to this circuit, there was a capacitance to ground at each node of at least 10FF in value. The SPLICE1.7 simulation required 4 min. while the SPICE2 simulation required 24 min. The memory requirements were 8 times less for the SPLICE1 job. The output waveforms are shown in Fig. 4.10.

Circuit Mosfets Diodes Nodes	uP Control Circuit 116 116 66	
	Time (s)	Memory (Kbyte)
SPICE2G	1426.8	205.9
SPLICE1.7	177.2	26.2
Ratios	8	8

Table 4.3

Comparison of conventional circuit simulation,

and ITA for an Industrial μP Control Circuit

4.4.4. Industrial 64K CMOS Static RAM Circuit

The block diagram for this example is given in Fig. 4.11 and each subcircuit schematic is shown in Fig 4.12. This circuit contained over 300 transistors and is an example of a industrial circuit which would be very expensive to simulate using SPICE2. The circuit contained only 36 explicit grounded capacitors out of 151 nodes. Diodes were used on the remainder of the nodes to model the parasitic junction capacitance effects. This was a sufficient condition to obtain convergence at every timepoint.

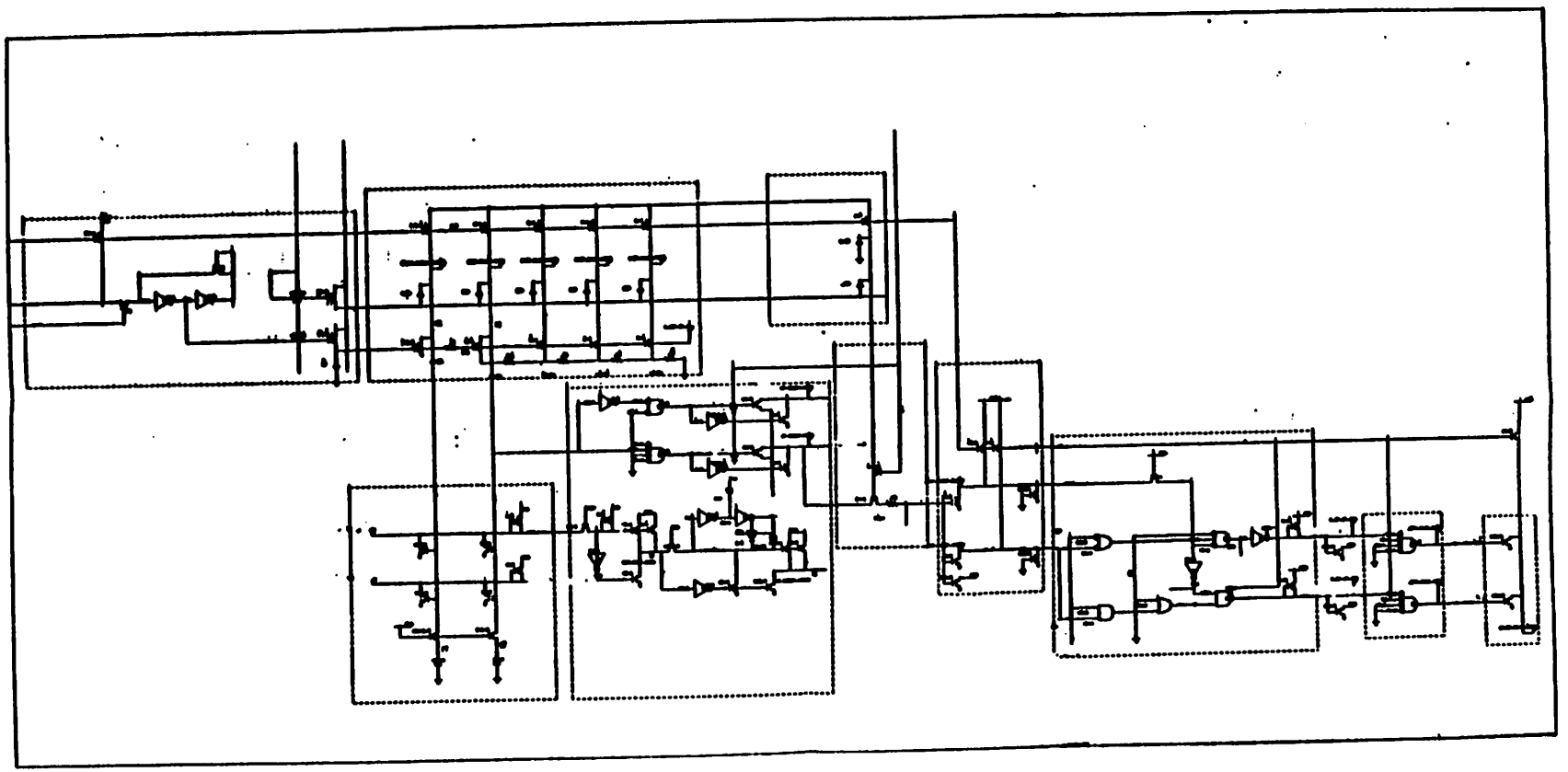


Fig. 4.9 : Worst-case path through an industrial Microprocessor Control Circuit

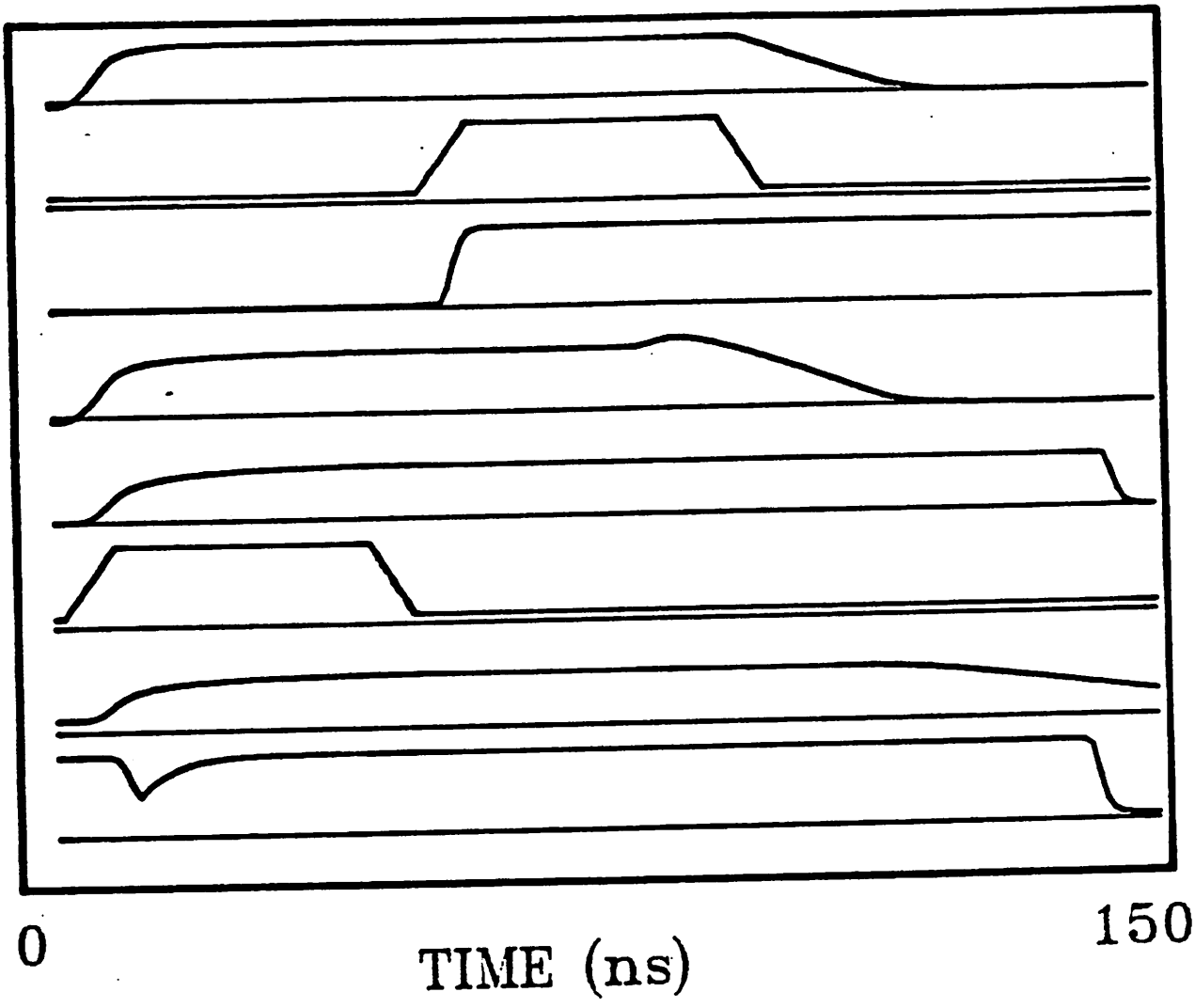


Fig. 4.10 : Output waveforms of uP Control Circuit

In this case, SPICE2 required approximately 3 hours to produce a solution whereas SPLICE1.7 required only 10 minutes. A comparison of the output waveforms is given in Fig. 4.13. The results are very close in all cases except in a few instances where SPICE2 exhibits point-to-point ringing. This is a product of the trapezoidal integration method used by default in SPICE2, which allows it to take larger timesteps but may cause ringing if the timestep is too large. SPLICE1.7 uses a Backward-Euler integration scheme, and for this method no numerical ringing is present in the waveforms.

Circuit Mosfets Diodes Nodes	64K CMOS SRAM 344 277 151	
	Time (s)	Memory (Kbyte)
SPICE2G	10446	506.3
SPLICE1.7	623	49.9
Ratios	16.75	10

Table 4.4

Comparison of conventional circuit simulation,

and ITA for an Industrial CMOS 64K SRAM

4.4.5. NMOS OpAmp Example

Although ITA was developed for the simulation of large digital circuits, the algorithm is robust enough to accurately simulate complex analog circuits. Therefore, an integrated circuit consisting mainly of digital circuitry along with a few analog blocks, typically found in telecommunication circuits and memory chips, can be simulated without any special precautions, other than the usual requirement of some grounded capacitance at every node.

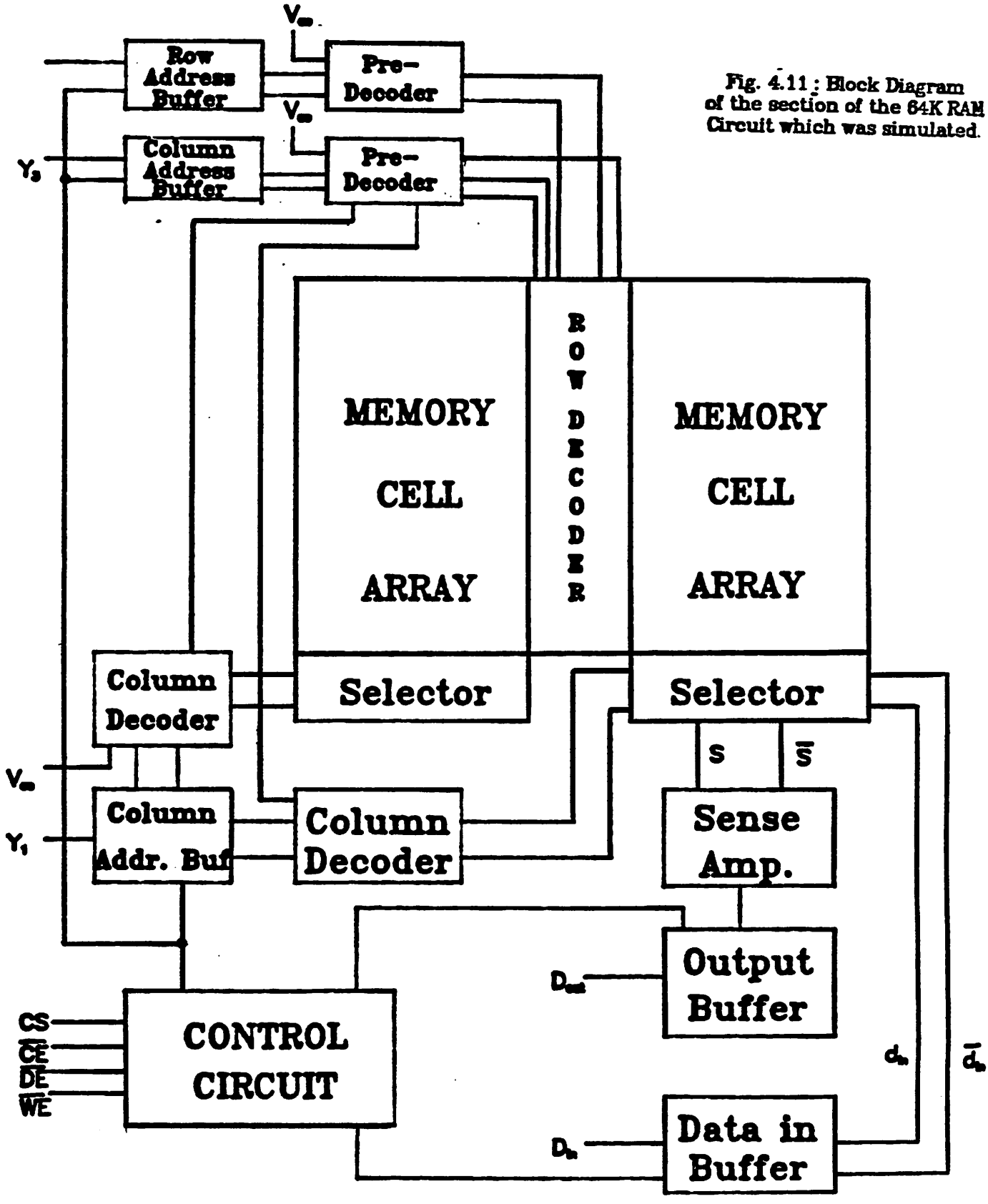


Fig. 4.11 : Block Diagram of the section of the 64K RAM Circuit which was simulated.

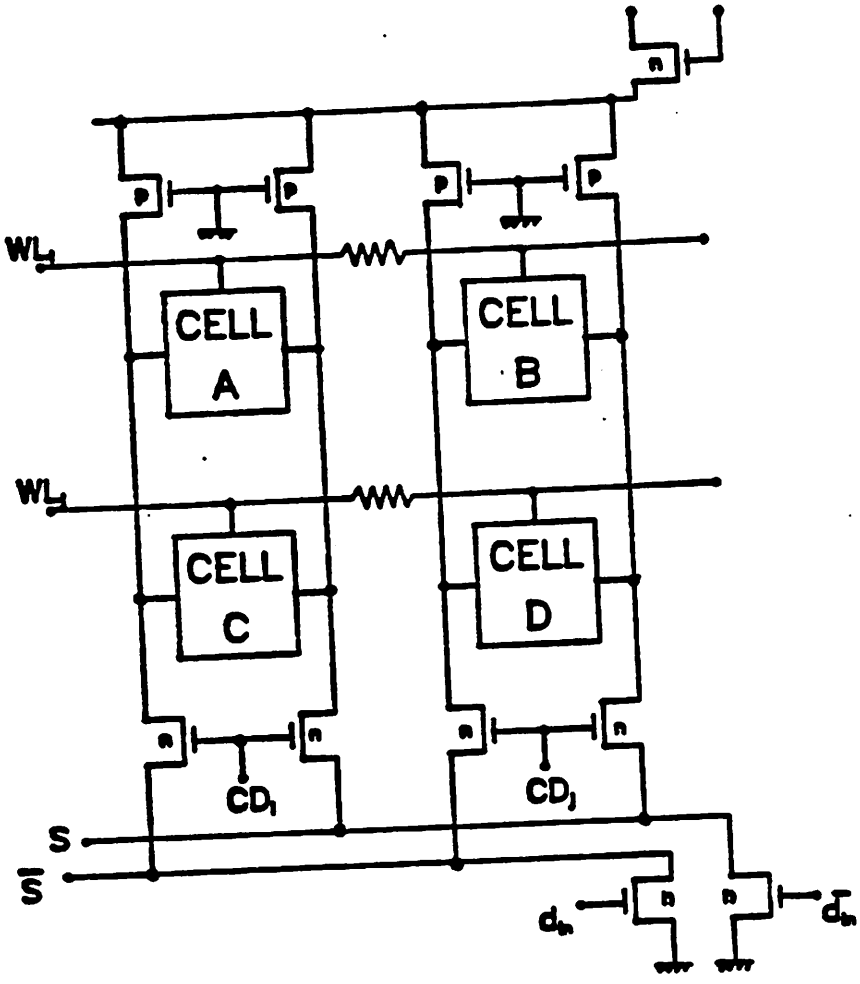


Fig. 4.12(a) : Memory Cell Array Details

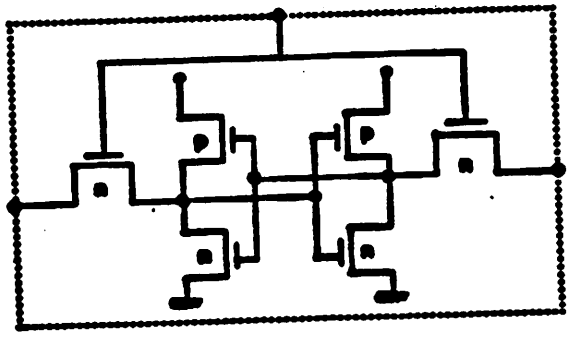


Fig. 4.12(b) : Memory Cell Box

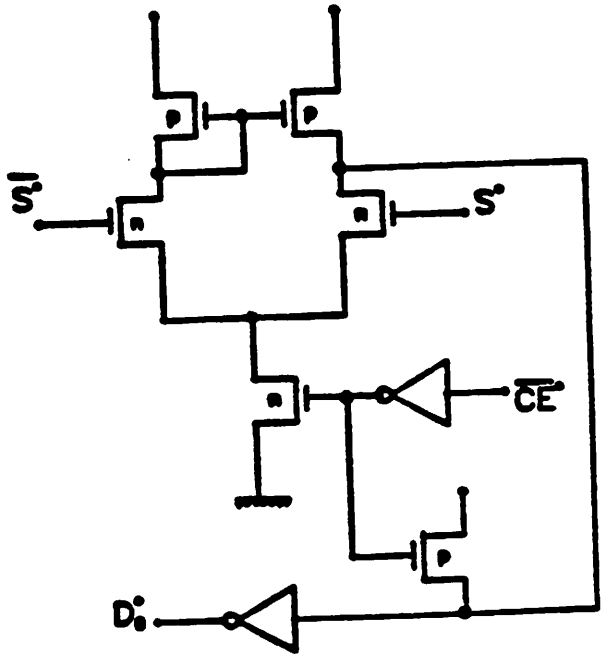


Fig. 4.12(c) : Sense Amplifier Circuit

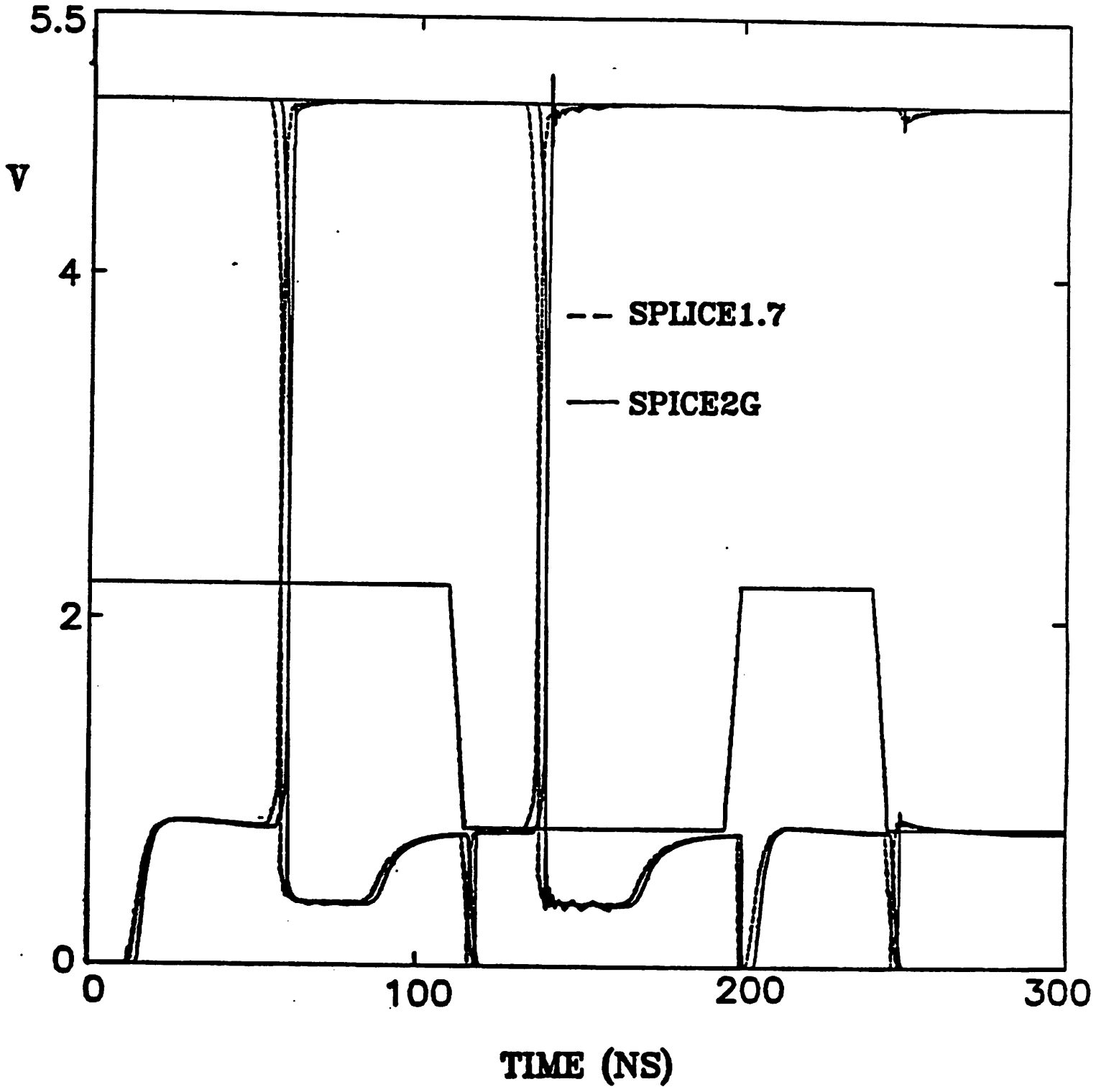


Fig. 4.13 : Output waveforms of 64K RAM circuit from SPLICE1 and SPICE2. Note the ringing produced in the SPICE2 output.

To illustrate the capability of the ITA method, the OpAmp in Fig. 4.14 was simulated using SPLICE1.7 and SPICE2. All the parasitic capacitances associated with each transistor were fully represented. As shown in the schematic diagram, there is a large 10pF compensation capacitor providing a capacitive feedback path in the circuit. The transistor at the output is $\frac{360\mu}{8\mu}$ to provide high gain at the output node. The circuit was connected in a unity-gain configuration and a step voltage was applied at the input. Fig. 4.15 is a comparison of the outputs of SPLICE1.7 and SPICE2. The results are identical except in the neighborhood of time t=0 due to slightly different initial conditions assumed by each program. However, the execution time of SPICE2 was two times faster than SPLICE1.7 because of the size and nature of the circuit. It is expected that this difference will be reduced as the program is developed further.

In general, ITA may be slower than SPICE2 when simulating small analog circuits because :

- they usually contain large feedback paths and high gain
- there is little or no latency in a typical analog circuit.

Therefore, the accuracy tolerances for the simulation (**abstol**, **reltol**) must be tight and the scheduling threshold, **mindvsch**, must be very small. There may also be a requirement for a small simulation timestep to guarantee convergence. Therefore, a dynamic timestep control mechanism is essential for the simulation of mixed analog/digital circuits to ensure that the global timestep will not be overconstrained by the analog circuits. Each node could have a local timestep which is based on its own Local Truncation Error estimation. Another useful feature would be to allow parameters such as **abstol**, **reltol** and **mindvsch** to be specified on a per-node basis. In this

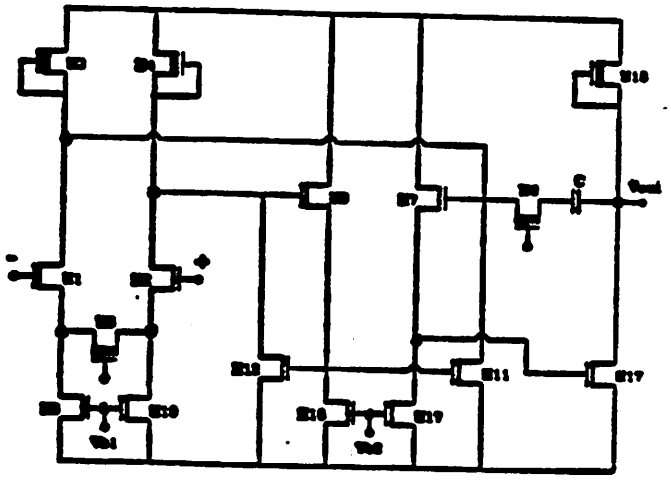


Fig. 4.14 : NMOS Operational Amplifier Circuit. This circuit features tight coupling between nodes and high forward gain (due to large output devices) and large capacitive feedback (due to the 10pF compensation capacitor).

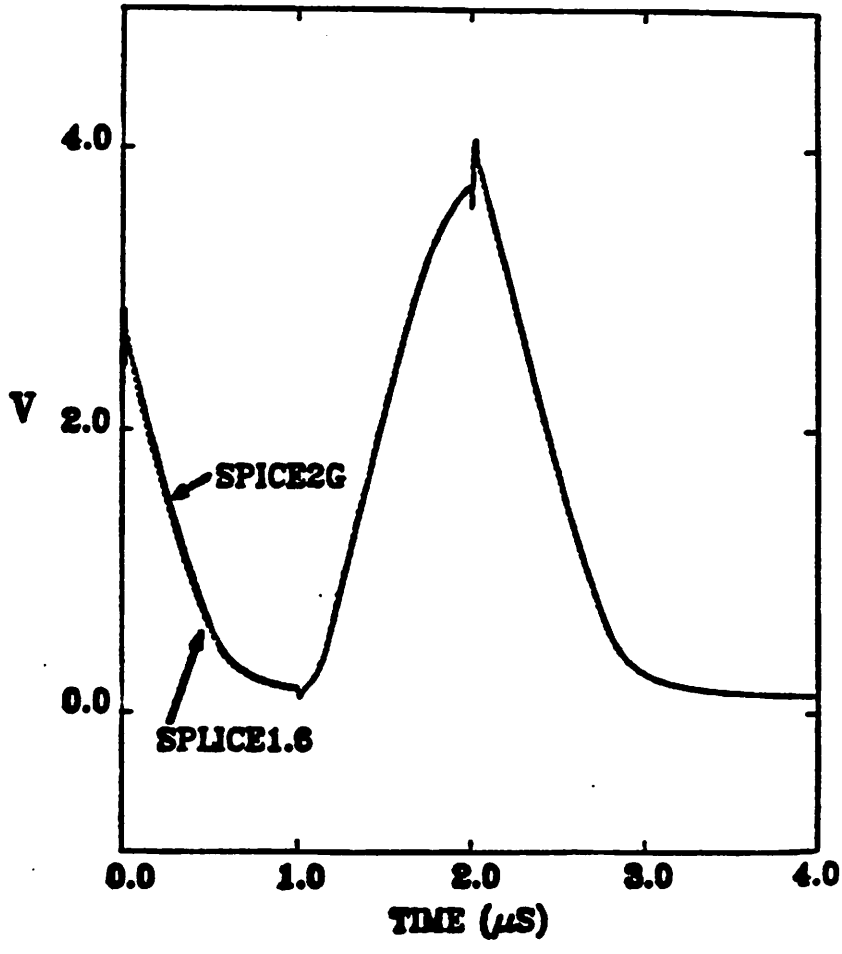


Fig. 4.15 : Output Waveforms of OpAmp circuit from SPLICE1 and SPICE2. The results are indistinguishable illustrating the accuracy of the ITA method.

way, certain nodes would be forced to iterate longer than others to ensure accuracy at these nodes. These and other techniques may be used to improve the performance of the simulator for handling analog circuits, although ITA is not ideally suited to the task.

4.4.6. CPU-time vs. Circuit Size

The run-times for the circuits described in this section are plotted against the circuit size in Fig. 4.16. It is clear from this plot that SPLICE1.7 is much faster than the SPICE2 program for large circuits. In fact, as the circuit size increases, the improvement factor increases. This is due to the fact that the linear equation solution time in SPICE2 increases rapidly with the circuit size, as described in Chap. 2. The run-time in SPLICE1.7 is proportional in the activity in the circuit and the **mrt** rather than circuit size. If the circuit is small, the standard approach is usually more efficient.

4.5. Mixed-Mode Examples

To complete this section, a pair of examples are presented using a logic/switch combination and a logic/electrical combination. The circuit to be simulated is a CMOS 4x5 multiplier with a 4-bit counter to generate a test sequence, as shown in block form in Figs. 4.17(a) and 4.17(b). The entire circuit is shown in Fig. 4.18. The multiplier uses the novel adder circuit illustrated in Fig. 4.19(a) and the exclusive-OR circuit of Fig 4.19(b). It is clear from this figure that the adder would be difficult to represent using Boolean gates. Therefore, a switch-level description is appropriate. For the simulation, the multiplier was connected in a multiply-by-2 configuration by setting the B-bits to 2. The counter, shown earlier in conjunction with the CDE cir-

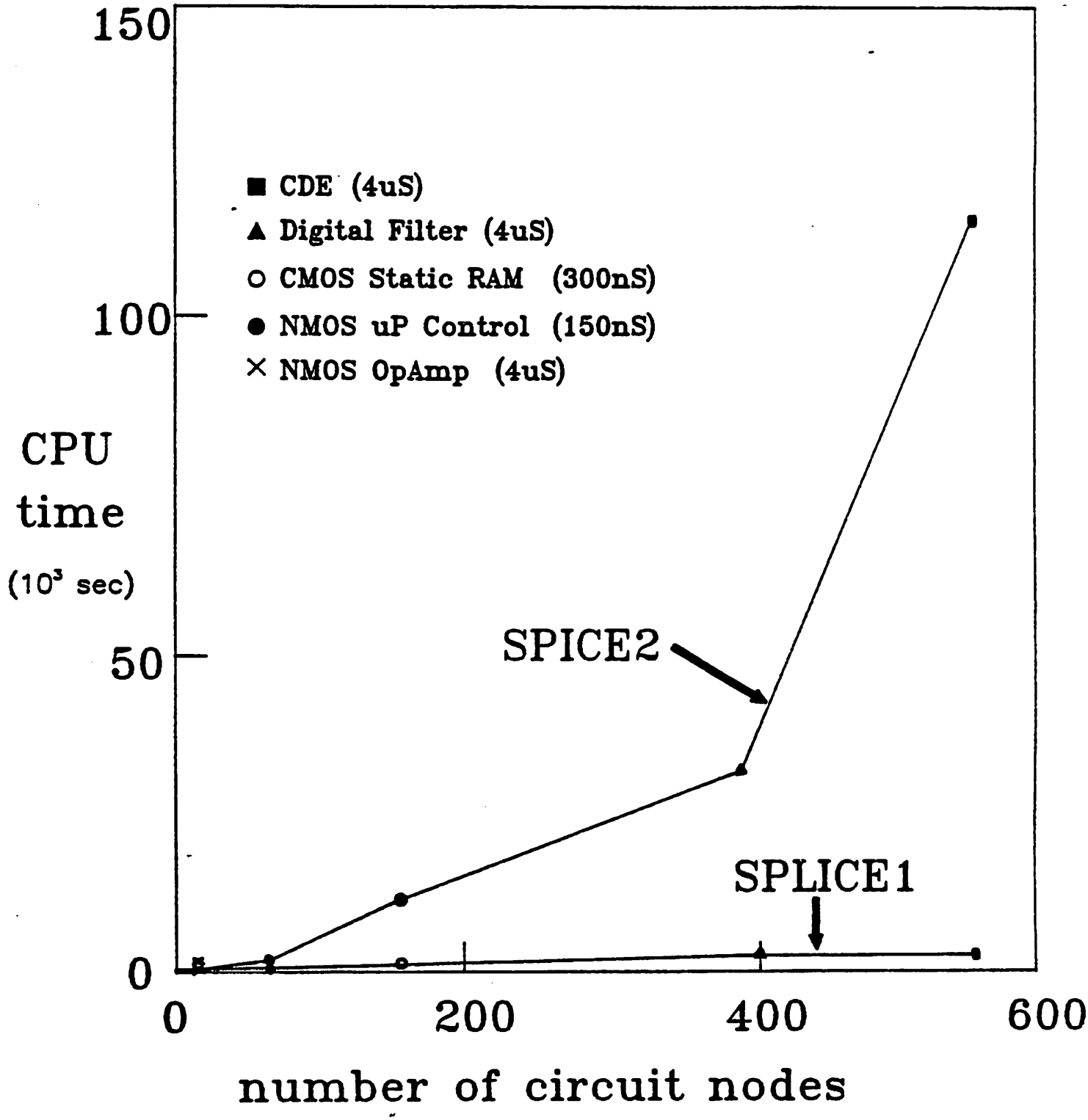


Fig. 4.16 : A plot of the results obtained using SPLICE1 and SPICE2.

cuit, was represented at the logic level using boolean gates. It provided a test sequence which was applied to the A-bits.

Initially, the operation of the multiplier was verified at the switch-level. This required only 7.9 CPU-seconds, *including the simulation of the counter circuit*. The output of this simulation is given in Fig. 4.20(a). Note that the outputs, p0, p1, p2 and p3, are evaluated at the input edges, since the simulator is operating in zero-delay mode at the switch-level.

After debugging the circuit at the switch-level, the description of the multiplier was changed from a switch-level description to an electrical-level description by simply changing the underlying models associated with the transistors. The majority of the description was left unchanged. The counter circuit description at the logic level was left in the circuit to generate the test inputs for the A-bits, as before. Logic-to-Voltage converters were inserted where necessary. This simulation required 682.7 seconds. The output of the simulation is shown in Fig. 4.20(b).

Some useful ways to use the mixed-mode capability in SPLICE1 have been illustrated in this example:

(1) One can debug the circuit very efficiently using zero-delay switch-level simulation. Then, a more detailed simulation can be performed to determine exact delays at the electrical level with very few changes in the circuit description, if the design is described hierarchically.

(2) A complicated logic circuit can be used to generate test inputs for an electrical simulation as opposed to using logic sources as input. In fact, a master clock signal was the only input waveform for the mixed-mode simulations described here.

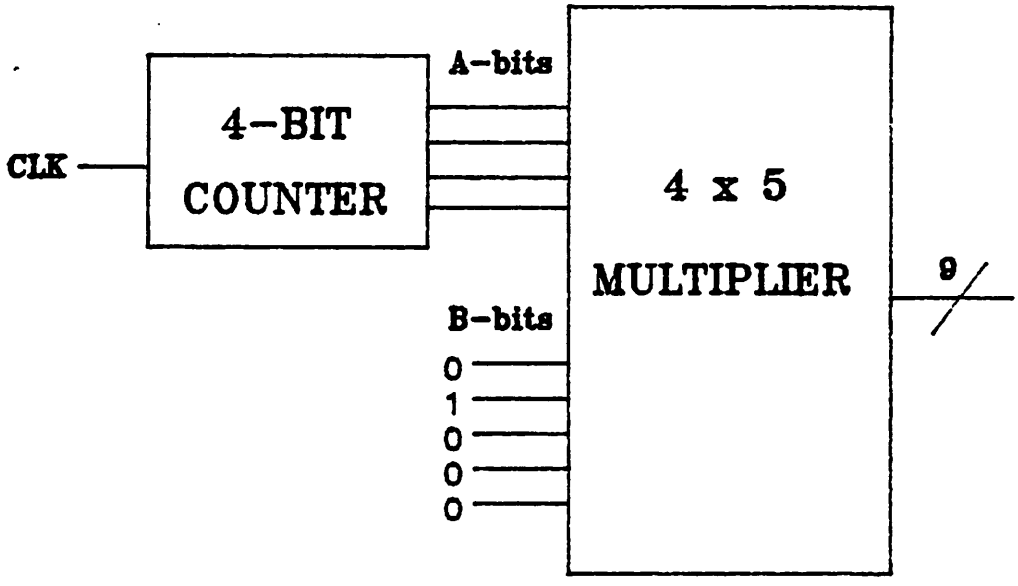


Fig. 4.17 : Block diagram of 4x5 Multiplier Circuit with Counter Circuit

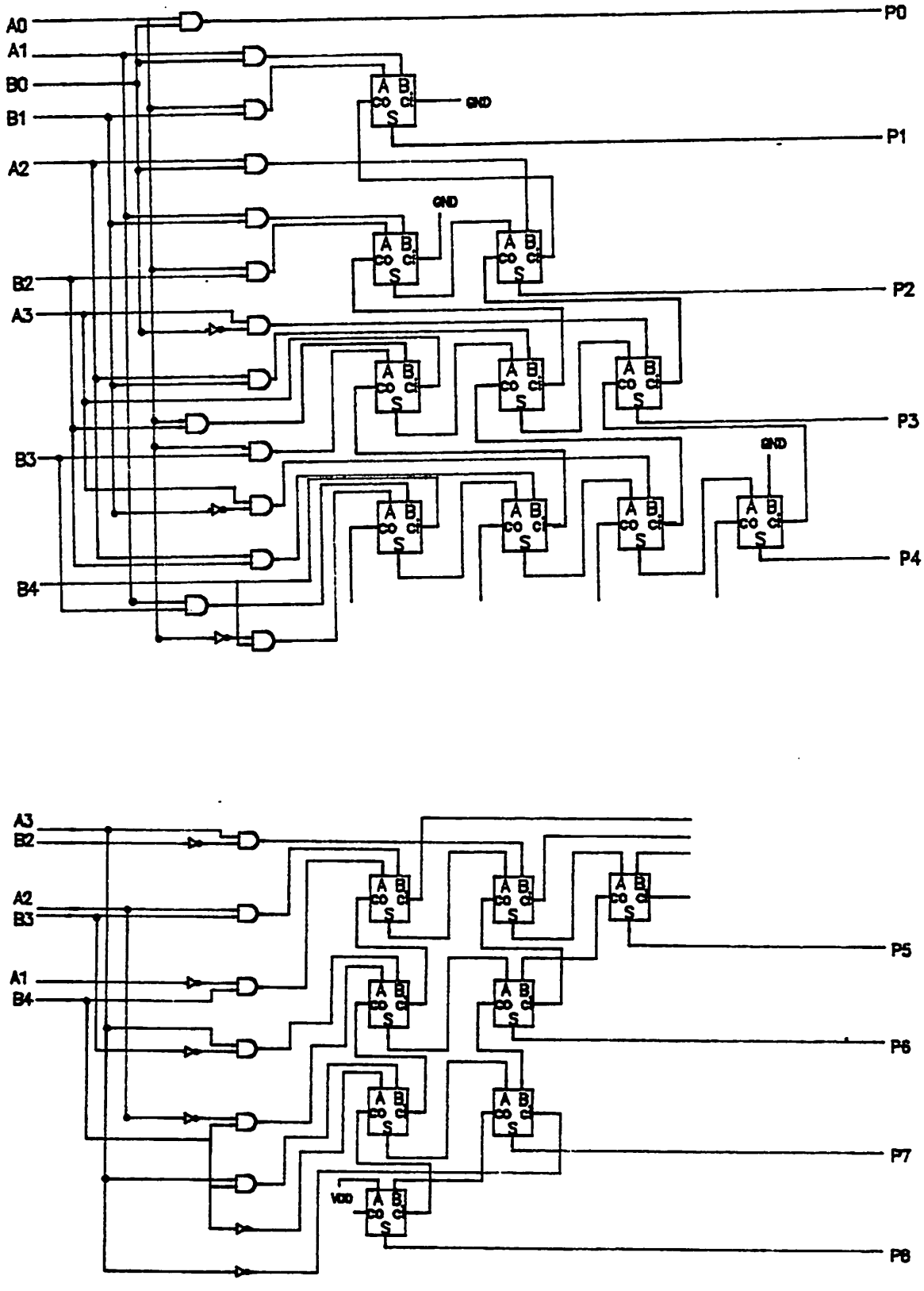
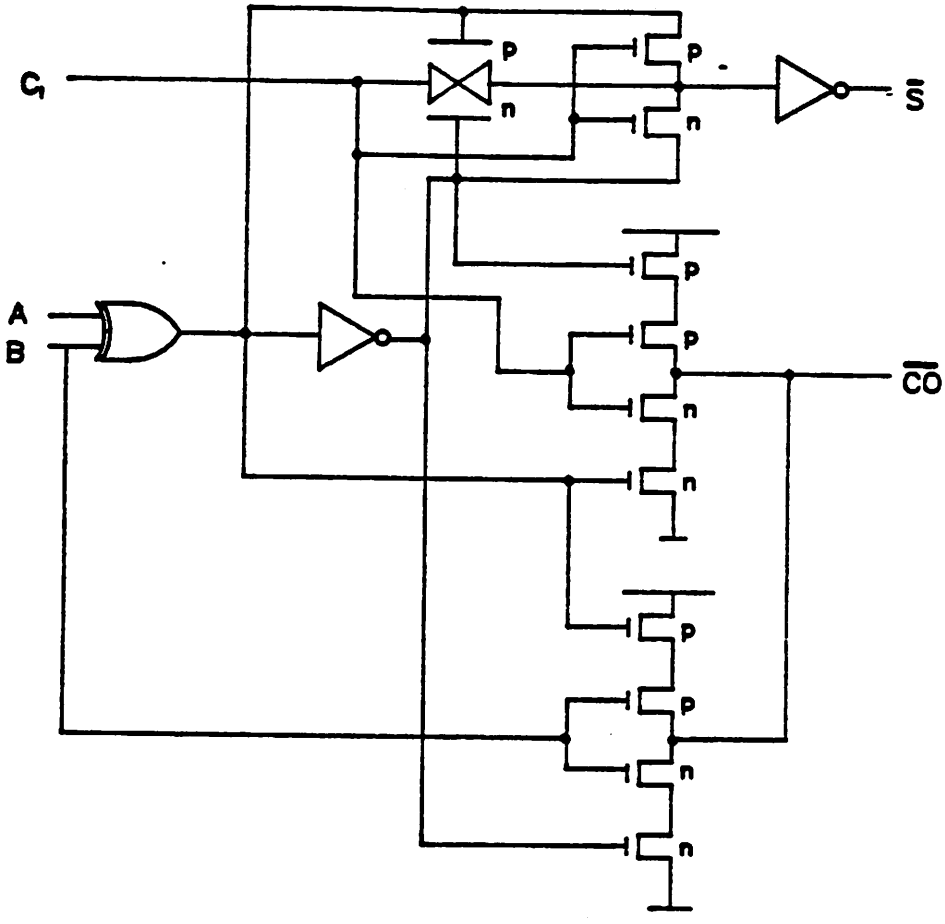


Fig. 4.18 : Details of 4x5 Multiplier circuit



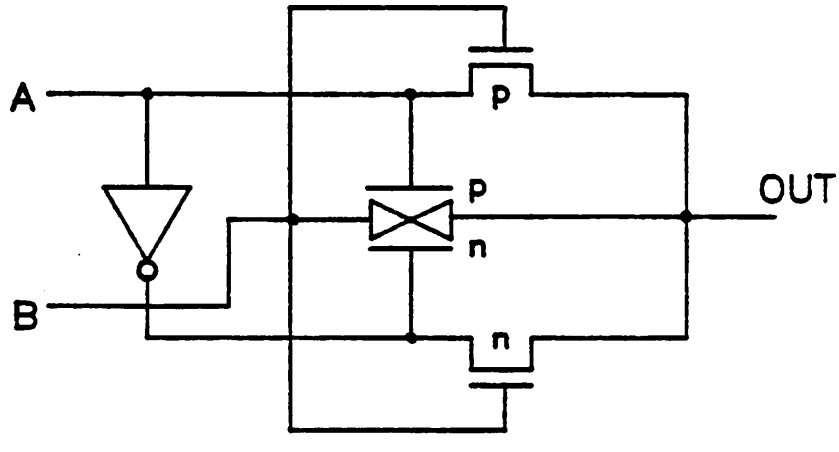


Fig. 4.19 : Details of Adder and Exclusive-OR circuits

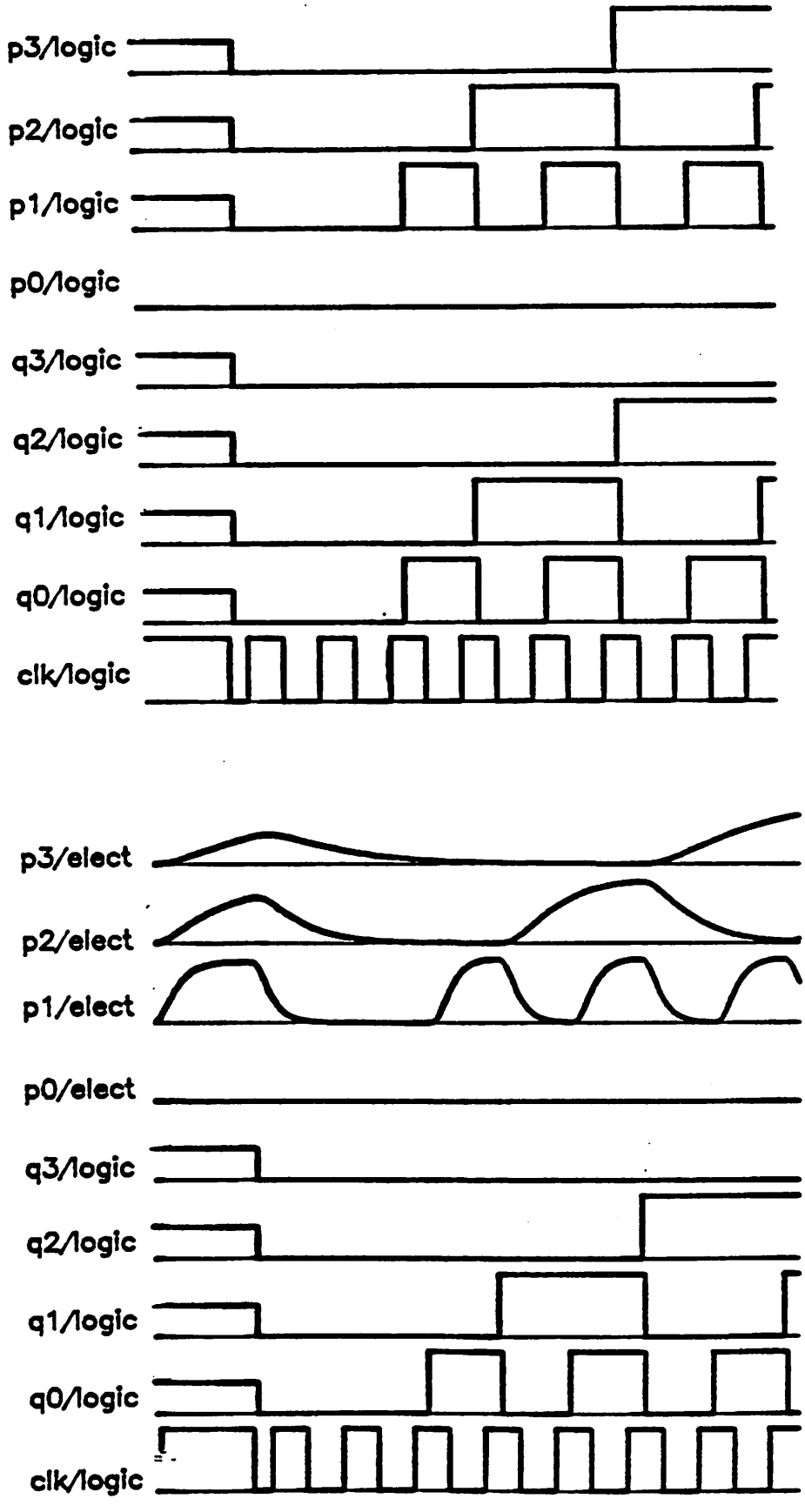


Fig. 4.20 : Output of Switch-level and Electrical-level Simulations

(3) A complicated circuit can be decomposed into small blocks and each block can be simulated with ITA electrical simulation. Once each block has been checked, a switch-level model can be generated which matches the logic characteristics of the cells. These blocks can then be combined for a switch-level analysis of the entire circuit which is relatively inexpensive compared to electrical simulation.

The results of the simulations are summarized in the table below.

Circuit	4x5 Multiplier	
Mosfets	545	
Multiplier Nodes	248	
Counter Gates	124	
Counter Nodes	130	
	Time (s)	Memory (Kbyte)
Switch-level	7.9	64.5
Electrical-level (ITA)	682.7	68.3

Table 4.5

Mixed-Mode simulation results

CHAPTER 5

5. CONCLUSIONS

SPLICE1 has been greatly improved by incorporating the new techniques described in this report. As evidenced by the statistics in Chap. 4, the new electrical simulation approach, ITA, is substantially faster than SPICE2 and requires far less storage. This method has shown so much promise that efforts are underway to generalize it as a standard circuit simulation approach. As pointed out earlier, the major problem with the method is the number of iterations required to obtain a solution when floating capacitors are present in the circuit. As the prototype program is developed further, it is expected that the performance characteristics will be significantly better than SPICE2. The ITA method provides a way to efficiently simulate large digital circuits and it may replace the standard approach in this application. It is also suitable for implementation on special-purpose hardware and work is underway in this area. Other areas of future work include the extension of the method to use Modified Nodal Analysis, dynamic timestep control and error control mechanisms.

The logic analysis in SPLICE1.7 has been enhanced to perform true-value logic simulation using a strength-oriented MOS model. This not only allows accurate modeling at the logic level but also provides a mechanism to perform accurate mixed-mode simulation. There is still work to be done in the area of strength modeling for logic elements to define the electrical/logic interfaces more accurately. SPLICE1.7 handles logic transfer gates in a consistent manner but the CNR method is not appropriate for a multiprocessor architecture. There is also the issue of delay modeling at the switch-level

which has not been addressed here. Research is currently being directed at applying multiple iterations at the logic level to determine state and delay information in transistor-level logic circuits.

In conclusion, the concepts presented in this report suggest that consistent electrical and logic simulation can be performed at the transistor-level using relaxation-based algorithms and event-driven selective trace techniques.

References

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

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

3. A.R. Newton, "Timing, Logic and Mixed-mode Simulation for Large MOS Integrated Circuits," pp. 175-240 in *Computer Design Aids for VLSI Circuits*, ed. P. Antognetti, D.O. Pederson, and H. De Man,Sijithoff and Noordhoff (1981).

4. 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).

5. S.P. Fan, M.Y. Hsueh, A.R. Newton , and D.O. Pederson, "MOTIS-C : A New Circuit Simulator for MOS LSI Circuits," *Proc. IEEE Int. Symp. on Circ. and Sys.* , (April 1977).

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

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

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

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

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

11. 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).

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

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

14. N. Tanabe, H. Nakamura, and K. Kawakita, "An MOS Circuit Simulator for LSI," *Proc. IEEE Int. Symp. on Circ. and Sys.*, pp. 1035-1039 (April 1980).

15. G.R. Boyle, "Simulation of Integrated Injection Logic," *ERL Memo. UCB/ERL M78/13*, University of California, (March 1978). Ph.D. Dissertation.

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

17. K. Sakallah and S.W. Director, "An Activity-Directed Circuit Simulation Algorithm," *Proc. IEEE Int. Conf. on Circ. and Computers*, (October 1980).

18. E. Cohen, "Performance Limits of Integrated Circuit Simulation on a Dedicated Minicomputer System," *ERL Memo. UCB/ERL M81/29*, (May 1981). Ph.D. Disseration.

19. A. Vladimirescu and D.O. Pederson, "Performance Limits of the CLASSIE Circuit Simulation Program," *Proceedings of the Int. Symp. on Circ. and Syst.*, (May 1982).

20. E. Lelarasmee, A. Ruheli , and A.L. Sangiovanni Vincentelli, "The Waveform Relaxation Method for the Time-Domain Analysis of Large Scale Integrated Circuits," *IEEE Tran. on CAD of Int. Circ. and Sys.* **Vol CAD 1, No. 3** pp. 131-145 (Aug 82).

21. 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).

22. 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).

23. R. A. Saleh, J. E. Kleckner, and A. R. Newton, "Iterated Timing Analysis and SPLICE1.6," *Proc. IEEE Int. Conf. on Computer-Aided Design*, (September 1983).

24. L.O. Chua and P.M. Lin, *Computer-Aided Analysis of Electronic Circuits: Algorithms & Computational Techniques*, Prentice-Hall, Inc., Englewood Cliffs, N.J. (1970).

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

26. 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).

27. J.E. Kleckner, *Iterated Timing Analysis and SPLICE2*, To be published

28. G.R. Case, "The SALOGS - A CDC 6600 Program to Simulate Digital Logic Networks," Sandia Laboratory Report No. SAND 74-044 (1975).

29. D. Dumlugol, H. De Man, P. Stevens, and G. Schrooten , "Local Relaxation Algorithms for Event Driven Simulation of MOS Networks Including Assignable Delay Modelling," *IEEE Trans. on CAD of Integrated Circuits*, (July 1983).

30. Graeme Boyle, Private communication.

31. Gregory D. Jordan and Ravi M. Apte, "Modeling of MOS Transistors in a Logic Simulator," *Proc. IEEE Int. Conf. on Circ. and Comp.*, pp. 431-434 (October 1982).

32. C.J. Terman, "Simulation Tools for Digital LSI Design," *Proposal for Ph.D. Research*, Massachusetts Institute of Technology, (December 1981).

33. *The TTL Data Book for Design Engineers - 2nd Edition*, Texas Instrument Incorporated (1976). See flipflop(7474B p.76), counter(74163 p.326), decoder(74154 p.309), encoder(74148 p.291).

34. Jim Kleckner constructed the CDE circuit.

35. D. Senderowicz, "An NMOS Integrated Vector-Locked Loop," *Memo. No. UCB/ERL M82/32*, University of California, (Nov. 1982).

36. S. Kuninobo designed the ADDER circuit.

APPENDIX I

Input Files for Example Circuits

The example circuits may be obtained from the University of California at Berkeley

APPENDIX II

SPLICE1.6 Data Structures

APPENDIX II

SPLICE1.6 Data Structures

(1) **Nodes:** The node data structure is set up in GENFS for the logic, electrical and vrail nodes.

LOGIC NODE :

offset	abbrev.	definition
0	fop	fanout pointer
1	fip	fanin pointer
2	type	=1 (for logic node) =-1 (for logic output node)
3	ts*	fanout schedule time
4	lval	logic value (3-bits for current value b2b1b0 3-bits for previous value b5b4b3 1=0, 2=1, 3=X)
5	lstr	logic strength (16-bits for current value 16-bits for previous value minimum strength = 1 ; maximum strength = 65,536
6	modptr	1: capacitance at node 2: node decay delay value
7	dectim	node decay time

ELECTRICAL NODE :

offset	abbrev.	definition
0	fop	fanout pointer
1	fip	fanin pointer
2	type	= 2 (for electrical node) =-2 (for electrical output node)
3	ts*	fanout schedule time(last time or next time)
4	Vn-1	current node voltage
5	Vn-2	previous node voltage
6	capptrs	points to node capacitance values in rvals
7	tsn-1*	last time processed (associated with Vn-1)
8	tsn-2*	previous time processed (associated with Vn-2)

VRAIL NODE :

offset	abbrev.	definition
0	fop	-1 (not used)
1	fip	-1 (not used)
2	type	=5 for a vrail node
3	vn	current node voltage = constant
4	vn-1	previous node voltage = constant = vn

INTEGER information is typically accessed using the nodptr array

i.e. info = imem(nodptr+locnod+ipos)

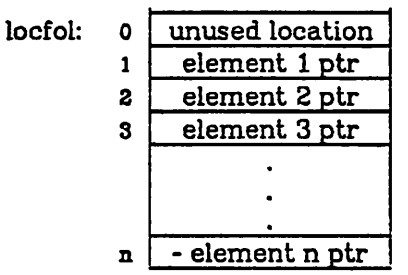
imem : integer memory maintained by memory manager
nodptr : node information data structure origin
locnod : position of 1st piece of info for node
ipos : position of desired info

REAL information is accessed through one more level of indirection:

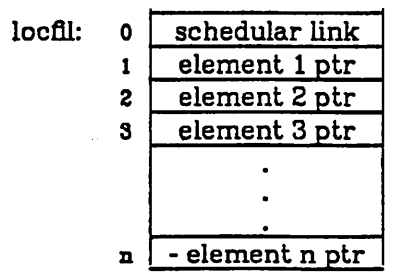
i.e. capacitance = rmem(rvals+imem(nodptr+locnod+5))

rmem : real memory maintained by memory manager
rvals : origin of real value array

(2) **Fanin and Fanout Lists**: Fanin and fanout lists are stored with the node data structure. Fanins to a node are all elements which can affect the value of the node. Fanouts of a node are all elements which can be affected by a new value at the node. They are set up in the LOGFA, TIMFA and ENDFA subroutines.

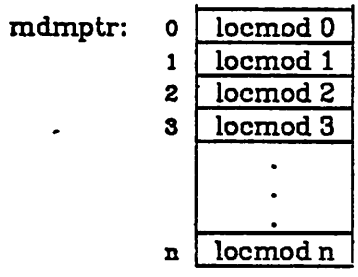


If there is only one element in the fanin list (which is often the case), then this list does not exist. The fil pointer in the node data structure has a -ve sign to denote that it is the element pointer itself.

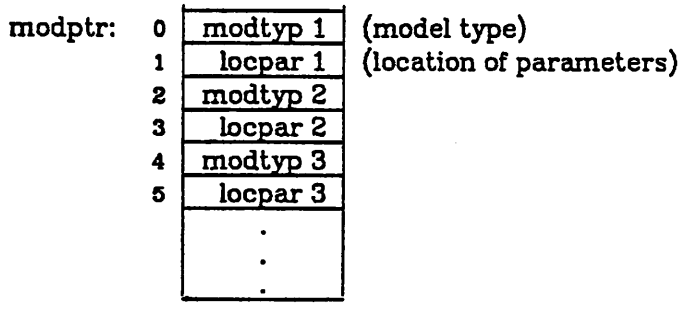


(3) **Models**: SPLICE1 stores model information using two levels of indirection so that one model may be referenced by many elements.

model info pointers are stored in an array called mdmptr:



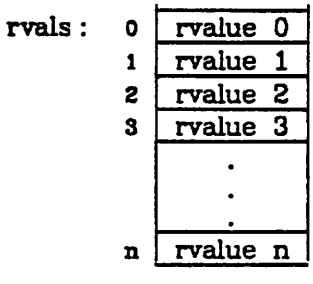
locmod points into a table called modptr which is organized as follows:



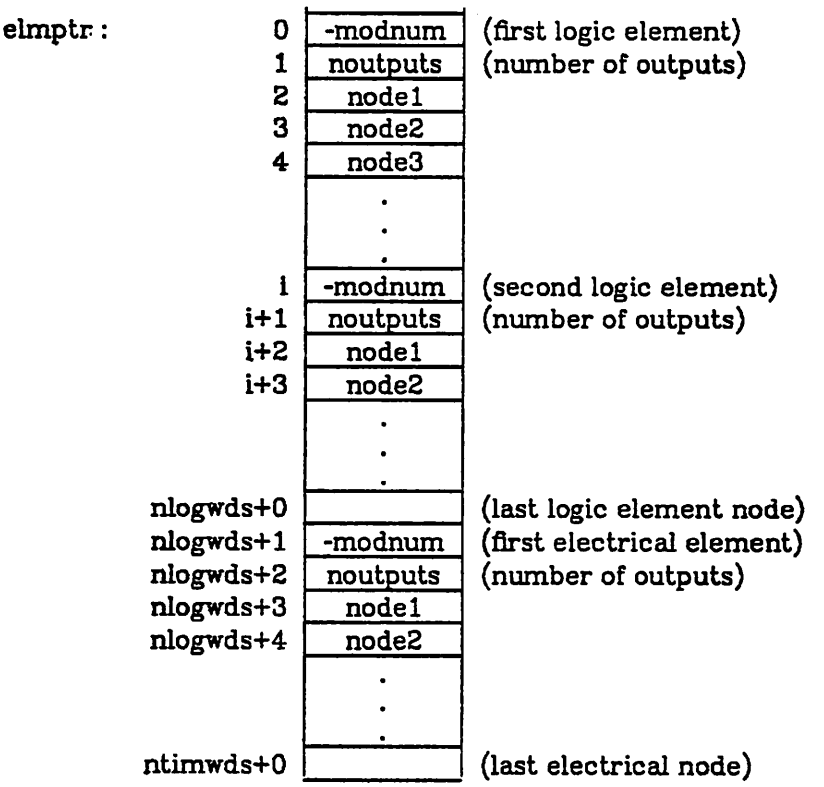
locpar points into rvals which is an array of floating-point quantities and so parameters are accessed as follows:

parameter = rmem (rvals + locpar)

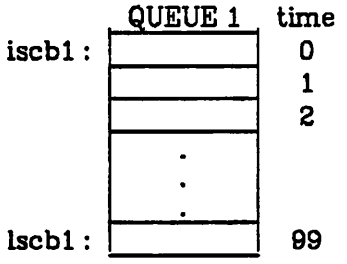
The rvals array is just a set of real values in the rmem space.

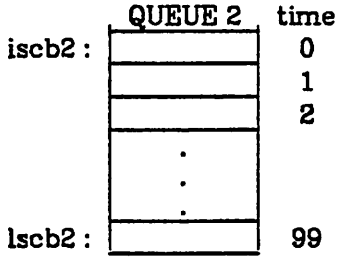


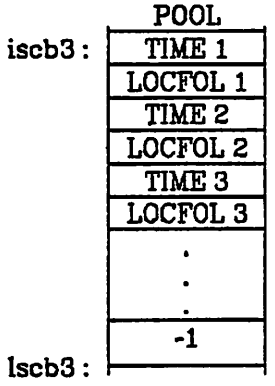
(4) **Elements:** Elements are initially written out to scratch files (timel, logel) by the routine SAVEL. Once they are read back in, they are stored in the array elmptr with the following format:



(5) **Schedular**: The time queue is made up of 2 - 100 word arrays and a pool for any events which do not fall within 200 timepoints of the beginning of the queue.







APPENDIX III

SPLICE1.6 Electrical Element Model Equations

Electrical Element Model Equations

1. Resistors

$$G_{eq} = \frac{1}{R}$$

$$I_{eq} = \frac{(V_1 - V_2)}{R}$$

2. Floating Capacitors

$$G_{eq} = \frac{C_{float}}{h}$$

$$I_{eq} = C_{float} \frac{(V_1^n - V_1^{n-1}) - (V_2^n - V_2^{n-1})}{h}$$

3. Transistors

a. Triode Region

$$I_{eq} = \mu C_{ox} \frac{W}{L}(V_{gs} - V_T - \frac{V_{ds}}{2}) V_{ds}(1.0 + \lambda V_{ds})$$

Drain node

$$G_{eq} = \mu C_{ox} \frac{W}{L}((V_{gs} - V_T - \frac{V_{ds}}{2}) V_{ds}\lambda + (1.0 + \lambda V_{ds})(V_{gs} - V_T - V_{ds}))$$

Source Node

$$G_{eq} = \mu C_{ox} \frac{W}{L}((V_{gs} - V_T + V_{ds} \frac{\gamma}{\sqrt{V_{sb} + 2\varphi_F}})(1.0 + \lambda V_{ds}) + (V_{gs} - V_T - \frac{V_{ds}}{2})\lambda V_{ds})$$

b. Saturation Region

$$I_{eq} = \frac{\mu C_{ox}}{2} \frac{W}{L}(1.0 + \lambda V_{ds})(V_{gs} - V_T)^2$$

Drain Node

$$G_{eq} = \frac{\mu C_{ox}}{2}\frac{W}{L}(V_{gs}-V_T)^2\lambda$$

Source Node

$$G_{eq} = \frac{\mu C_{ox}}{2}\frac{W}{L}((V_{gs}-V_T)^2\lambda+(V_{gs}-V_T)(1.0+\frac{\gamma}{\sqrt{V_{sb}+2\varphi_F}})(1.0+\lambda V_{ds}))$$

APPENDIX IV

Source Code for SPLICE1

The SPLICE1 program is available in the public domain from the University of California at Berkeley