THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 9, November 1981
Printed in U.S.A.

Fault-Simulation Methods—Extensions and Comparison

By Y. H. LEVENDEL and P. R. MENON

(Manuscript received December 16, 1980)

In this paper, we compare four different methods of fault simulation in terms of their handling of arbitrary numbers of logic values, modeling levels, and detailed timing. The methods considered are parallel, deductive, multilist, and concurrent simulation methods. Since some of the methods, in their current forms, are unable to handle all the problems under consideration, we have proposed extensions to the methods wherever necessary before making the comparisons. While all the methods considered are capable of solving the problems with the same degree of accuracy, the concurrent simulation method appears to be the simplest and most flexible.

I. INTRODUCTION

Different techniques for the efficient simulation of faults in digital circuits have been published. Among these, the best known are parallel simulation,[1-3] deductive simulation,[4] and concurrent simulation.[5] A few papers analyzing some aspects of these methods have also been published.[6-9]

This paper and two others[10,11] comprise a series attempting a comprehensive analysis of fault simulation methods. It is hoped that they will provide a basis for the selection of fault-simulation methods to satisfy specific requirements.

In this paper, we consider three aspects of circuit modeling and their effects on the fault-simulation method used. First, we consider the number of logic values needed to accurately model logic devices and its impact on the simulation method. Next, the effectiveness of the different methods for simulating at different levels (e.g., gate level, functional level, subsystem level, etc.) is considered. Finally, we discuss the modeling of timing effects, such as rise and fall times and high-frequency rejection.

Our study covers four methods of fault simulation: parallel, deductive, multilist, and concurrent. In their current forms, some of the methods are not capable of handling all the problems we consider.

Therefore, we have attempted to extend the existing methods, wherever necessary, before making the comparisons between methods. Before proceeding to the analysis of the methods, we present a brief description of each method.

Historically, parallel simulation was the first method that simulated a number of faults simultaneously.[1] This method, which is perhaps the most widely used, takes advantage of word-oriented operations in the host computer and packs together several faulty circuit values into one or more computer words. Although this method is quite efficient, multiple passes are required for simulating large numbers of faults.

Deductive simulation attempts to eliminate the need for multiple passes by computing normal signal values in the circuit and deducing the faulty values by manipulating lists of faults.[4] Associated with each signal is a fault list, which is a set of faults, any one of which will cause the signal value to be different from the normal value. The effects of faults are propagated through the circuit by an algebra of sets.

The multilist method associates two or more lists of faults with each signal.[10,12] Conceptually, the number of lists associated with a signal is equal to the number of logic values simulated. Thus, for two logic values, there will be a 0-list and a 1-list associated with each signal, the former being the set of faults in whose presence (individually) the signal will have the value 0, and the latter those that result in a value of 1. Set algebra is necessary for manipulating these lists also. However, unlike the deductive method, the equations for computing the output lists of a device from its input lists are dependent only on the function performed by the device and not on the signal values.

In concurrent simulation, any fault that causes the inputs, outputs, or internal state of a device to be different from their normal values is represented conceptually by a copy of the device. During simulation, if the inputs, outputs, and state of a faulty copy become identical to those of the fault-free copy, the faulty copy is deleted. Thus, faulty copies are created and deleted during simulation. The evaluation of faulty copies is essentially the same as fault-free copies, and no set algebra is involved. Concurrent simulation can also handle a large number of faults simultaneously.

It is interesting to note that all the above methods, except parallel simulation, use some form of data compression for storing faulty signal values. On the other hand, parallel simulation attempts to compute simultaneously the fault-free signal value and a number of faulty signal values associated with each lead in the circuit.

II. NUMBER OF LOGIC VALUES

Three-valued logic systems have been widely used for analyzing essentially binary systems.[13,14] Three logic values are also used in logic

simulation, where 0 and 1 represent the two discrete values being modeled and a third value, u, denotes that a particular value is unknown.

Recently, tri-state busing has become a widespread technique used in many LSI designs. Difficulties in modeling effects associated with CMOS technology have been reported.[15] One effect is the memory associated with a disabled bus. That is, the disabled bus remembers the previous logic value on the bus. A solution consists of adding special circuitry to regular gates, making possible the use of a simulator with only three logic values.[15] An alternate solution is the addition of three more logic values, namely z_0, z_1, and z, for representing the states of disabled buses, with previous value equal to 0, 1, and unknown, respectively.[16] Transistor-transistor logic (TTL) tri-state technology requires the addition of only one logic value, z.[16]

Bus contention, another typical, potentially destructive tri-state effect, cannot be modeled by added circuitry. A solution consists of adding one more logic value representing a conflict state, a, as shown in the following example.

Consider a driver inverter and a bus configuration in TTL tri-state technology (Fig. 1). When line e is enabled, the gate operates as an inverter, when e is disabled the output of the gate is in a high-impedance state. When used in a bus configuration, two enabled inverters create a conflict (bus contention), if they are in opposite states. The set of logic values $\{0, 1, u, a, z\}$ is sufficient to model these effects, since tri-state devices in TTL technology do not have the memory property mentioned above.

Table I shows how the bus configuration of Fig. 1 can be simulated using the above set of five logic values. Since the bus will be connected to the output of drivers, which can produce four out of the five logic values, only four logic values are used for modeling the bus.

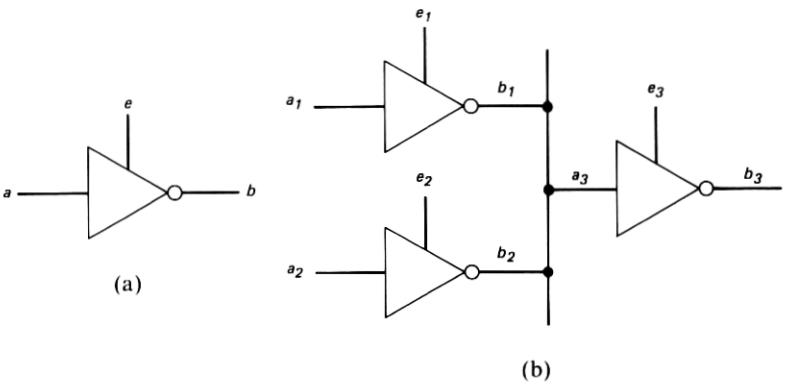

Fig. 1—(a) TTL Driver-inverter. (b) Bus configuration.

Table I—(a) Tristate inverter output
(b) State of tristate bus

a_i \ e_i	0	1	u	a	z
0	z	1	u	u	u
1	z	0	u	u	u
u	z	u	u	u	u
a	z	u	u	u	u
z	z	u	u	u	u

(a)

b_2 \ b_1	0	1	u	z
0	0	a	u	0
1	a	1	u	1
u	u	u	u	u
z	0	1	u	z

(b)

If an ordinary gate could be connected directly to a bus, the model should allow five logic values for the gate inputs, but requires only three logic values for its output. Table II shows the behavior of such an AND gate with inputs x and y, and output t.

The use of larger sets of logic values, though necessary to correctly model modern technology, has a serious impact on the method of simulation used. The following sections deal with this problem.

2.1 Parallel simulation

When using a switching algebra (i.e., two logic values) parallel simulation can be implemented by associating one computer word with each line in the circuit. One bit of this word represents the signal value on a line in the fault-free circuit and the remaining bits represent values on the same line in the presence of different single faults.

Table II—AND gate with five input logic values

y \ x	0	1	u	a	z
0	0	0	0	0	0
1	0	1	u	u	u
u	0	u	u	u	u
a	0	u	u	u	u
z	0	u	u	u	u

Table III—Coding for three logic values

α_i^0	α_i^1	a_i
0	0	unknown
0	1	1
1	0	0
1	1	unused

When a three-valued system is used, each of the circuits simulated in parallel must be coded using two binary digits. A commonly used method consists of associating two words with each line a, namely the 0-word, α^0, and the 1-word, α^1.[17] The coding used is shown in Table III, where the subscript i refers to the ith bit of each word.

Examples of its use are shown in Fig. 2. Here, and elsewhere in this paper, lower-case roman letters are used to denote leads and Greek letters represent words. For the gates of Fig. 2, we have

$$\begin{cases} \gamma^0 = \alpha^0 + \beta^0 \\ \gamma^1 = \alpha^1 \cdot \beta^1 \end{cases}$$

$$\begin{cases} \delta^0 = \alpha^0 \cdot \beta^0 \\ \delta^1 = \alpha^1 + \beta^1 \end{cases}$$

$$\begin{cases} \xi^0 = \alpha^1 \\ \xi^1 = \alpha^0 , \end{cases}$$

where $\cdot$ and $+$ represent the bitwise AND and OR operations on complete words.

This method can be extended for any number of logic values. For instance, consider the AND gate of Fig. 3, using the set of logic values $\{0, 1, u, a, z\}$. The binary coding scheme requires three computer words for each line, and three codes (out of eight) are not used. For

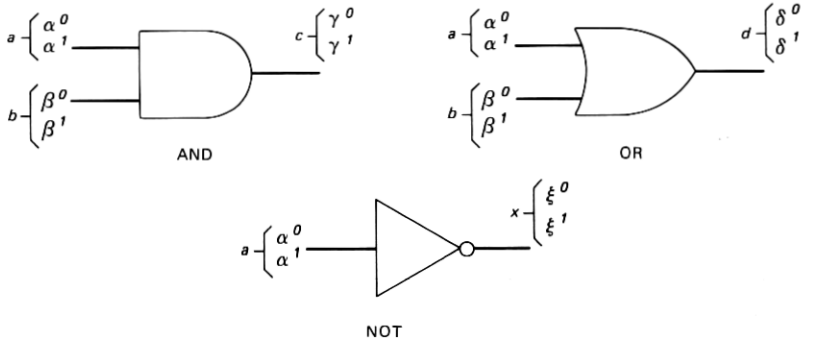

Fig. 2—Use of coding to represent signal values on gates.

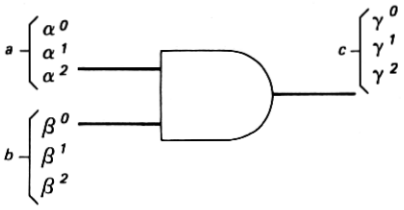

Fig. 3—AND gate representation for five logic values.

any choice of code, it is possible to calculate the gate output from switching expressions of the following form:

$$\gamma^0 = f(\alpha^0, \alpha^1, \alpha^2, \beta^0, \beta^1, \beta^2)$$
$$\gamma^1 = g(\alpha^0, \alpha^1, \alpha^2, \beta^0, \beta^1, \beta^2)$$
$$\gamma^2 = h(\alpha^0, \alpha^1, \alpha^2, \beta^0, \beta^1, \beta^2).$$

The original set of logic values and operations do not constitute a Boolean algebra. The coding scheme establishes a mapping of non-Boolean functions into switching operations that can be applied on full computer words, thus, allowing parallel processing.

By using a coding of $n - 1$ variables to represent n logic values, it is possible to obtain simpler equations for computing the outputs of gates. For example, consider the gate of Fig. 3 and a coding using four words α^0, α^1, α^a, and α^z to represent five logic values. The code is such that $\alpha_i^j = 1$, $j = 0, 1, a$, or z, iff $a_i = j$. All the variables will be zero, if and only if $a_i = u$. With this coding, the following equations are obtained for the gate of Fig. 3:

$$\gamma^0 = \alpha^0 + \beta^0$$
$$\gamma^1 = \alpha^1 \cdot \beta^1$$
$$\gamma^a = 0$$
$$\gamma^z = 0.$$

This type of coding can be used for any number of logic values.

2.2 Multilist simulation

It has been shown that for a three-valued logic system, three lists X^0, X^1, and X^u can be associated with each line x.[10,12] Each list, X^i, $(i = 0, 1, u)$ represents the faults which cause line x to have the value i. For each line x, all the lists X^i are disjoint and any list is the complement of the union of the other two (i.e., the union of the three lists is the set of all faults being simulated).

For the gates of Fig. 2, we have

$$\begin{cases} C^1 = A^1 \cap B^1 \\ C^0 = A^0 \cup B^0 \end{cases} \qquad C^u = \overline{C^0 \cup C^1}$$

$$\begin{cases} D^1 = A^1 \cup B^1 \\ D^0 = A^0 \cap B^0 \end{cases} \qquad D^u = \overline{D^0 \cup D^1}$$

$$\begin{cases} X^1 = A^0 \\ X^0 = A^1 \end{cases} \qquad X^u = \overline{X^0 \cup X^1},$$

where $^-$, $\cup$, and $\cap$, are set complement, union, and intersection, respectively.

When five logic values are used, we need five lists; for instance, A^0, A^1, A^u, A^a, and A^z are associated with line a.

For the AND gate of Fig. 3, we have

$$C^0 = A^0 \cup B^0$$

$$C^1 = A^1 \cap B^1$$

$$C^a = \{\ \}$$

$$C^z = \{\ \}$$

$$C^u = \overline{(C^0 \cup C^1 \cup C^z \cup C^a)} = \overline{A^0 \cup B^0 \cup A^1 \cap B^1}.$$

For the inverter of Fig. 1, we have

$$B^0 = E^1 \cap A^1$$

$$B^1 = E^1 \cap A^0$$

$$B^z = E^0$$

$$B^u = E^u \cup E^a \cup E^z \cup (E^1 \cap (\overline{A^0 \cup A^1}))$$

$$B^a = \{\ \},$$

and for the bus configuration of Fig. 1

$$A_3^0 = (B_1^0 \cap B_2^0) \cup (B_1^0 \cap B_2^z) \cup (B_2^0 \cap B_1^z)$$

$$A_3^1 = (B_1^1 \cap B_2^1) \cup (B_1^1 \cap B_2^z) \cup (B_2^1 \cap B_1^z)$$

$$A_3^u = B_1^u \cup B_2^u$$

$$A_3^z = B_1^z \cap B_2^z$$

$$A_3^a = (B_2^0 \cap B_1^1) \cup (B_2^1 \cap B_1^0).$$

This method can be generalized to any gate type and any number of logic values as follows: Let us assume that we wish to simulate a function $f(x_1, x_2, \cdots, x_n)$, where each input and the output may assume any one of k values, denoted by $1, 2, \cdots, k$, and that the

function is defined by a table which specifies the values of f for all combinations of values of x_i.

(*i*) We associate a variable x_i^j with each variable x_i, such that $x_i^j = 1$ if and only if $x_i = j$, $1 \leq j \leq k$. Similarly, we associate k variables f^j with f.

(*ii*) For each i, $1 \leq i \leq k$, we obtain an expression

$$f^i = \sum P_j,$$

where P_j are products of literals x_i^m, representing all combinations of values for which $f = i$. For example, if the table has an entry

$$x_1 = 1, x_2 = 0, x_3 = z, f = 1,$$

the expression for f^1 will contain the term

$$x_1^1 x_2^0 x_3^z.$$

(*iii*) Replace all lower-case letters in the equation for f^i by the corresponding upper-case letters, representing lists, and retain the superscripts and subscripts. Replace products by intersection and sums by union.

2.3 Deductive simulation

Deductive simulation is well defined for two logic values, and is also applicable to three logic values with some loss of information.[4] Specifically, if the signal value in the normal circuit is known, (i.e., 0 or 1), but the value in the presence of a fault α is unknown (denoted by u), the fault α is included in the fault list as a star fault;[4,18] that is, it is unknown whether the particular signal value in the presence of the fault α will be different from the fault-free value. It was shown in Refs. 10 and 12 that there are cases where the circuit value in the normal circuit may be unknown, but the value in the presence of a fault may be known. Since the deductive method cannot represent this case, the results obtained may be less accurate than with other methods.[10,12]

A modification of the deductive method that leads to accurate three-valued fault simulation was presented in Ref. 10. It uses the coding of Table III for representing each signal value by a pair of binary variables. A pair of equations can then be derived, as in Section 2.1, for computing the coded outputs for each gate type. These equations can be viewed as defining a transformation of the original circuit with three signal values into two circuits that will have only binary signals. These two circuits can be simulated using the two-valued deductive method. The fault-free and faulty signal values on any lead in the original circuit can be determined from the signal values and fault lists associated with the corresponding pair of leads in the transformed circuits.

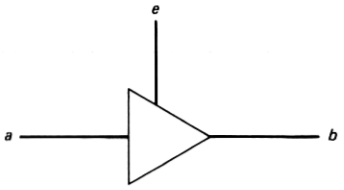

Fig. 4—Tristate bus driver.

The same approach can be used for performing deductive fault simulation with any number of logic values. If k logic values are to be simulated, $\lceil \log_2 k \rceil$ binary variables will be used to represent them, where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x. The equations for the coded outputs of different gate types can be derived from their truth tables, and used in deductive simulation.

As as example, consider the bus driver of Fig. 4 to be simulated with four logic values, namely, 0, 1, z (high impedance) and u (unknown). The behavior of the device is specified in Table IV.

Using the coding of Table V, we shall represent the signals a, e, and b of the bus driver by a_0 and a_1, e_0 and e_1, and b_0 and b_1. The output equations b_0 and b_1 can be derived from Tables IV and V.

$$b_0 = e_0 \cdot \bar{e}_1 + a_0 \cdot \bar{e}_0 \cdot e_1$$

$$b_1 = e_0 \cdot \bar{e}_1 + a_1 \cdot \bar{e}_0 \cdot e_1$$

For any combination of input values and fault lists, the output values and fault lists can be computed as in Ref. 19.

Denoting the fault list associated with each variable by the corresponding upper-case letter, let the input values and fault lists for the circuit of Fig. 4 be as follows:

$$a_0 = 0; \quad A_0 = \{1, 3\}$$

$$a_1 = 1; \quad A_1 = \{3\}$$

$$e_0 = 0; \quad E_0 = \{2, 4\}$$

$$e_1 = 1; \quad E_1 = \{4, 5\}$$

Table IV—Behavior of bus driver

		e			
		0	1	z	u
a	0	z	0	u	u
	1	z	1	u	u
	z	z	z	u	u
	u	z	u	u	u

Table V—Coding for tristate devices

x_0	x_1	x
0	0	u
0	1	1
1	0	0
1	1	z

Let us assume that all the faults being considered are external to the device, and we wish to propagate the effects of the faults through the device. The input conditions are: $a = 1$, $e = 1$. Since the fault 1 is contained only in the fault list A_0, it will cause a_0 to become 1, and therefore a to become z. On the other hand, fault 3 is contained both in A_0 and A_1, and will cause both a_0 and a_1 to be inverted; the value of a in the presence of fault 3 will be 0. Similarly, fault 2 will result in $e = z$, fault 4 in $e = 0$, and fault 5 in $e = u$.

For the above set of values, the output values and fault lists can be computed using the equations for b_0 and b_1 and the method presented in Ref. 19 as follows:

$$b_0 = 0$$

$$b_1 = 1$$

$$B_0 = (A_0 \cap \overline{E_0} \cap \overline{E_1}) \cup (E_0 \cap E_1) = \{1, 3, 4\}$$

$$B_1 = (A_1 \cup E_0 \cup E_1) \cap \overline{(E_0 \cap E_1)} = \{2, 3, 5\}.$$

Denoting the value of b in the presence of fault α by $b(\alpha)$, we can obtain the following faulty values from the values b_0 and b_1 and fault lists B_0 and B_1.

$$b(1) = z; \quad b(2) = u; \quad b(3) = 0; \quad b(4) = z; \quad b(5) = u.$$

These can be verified by computing the output for each faulty combination of inputs using Table IV.

The modified deductive method discussed above does not lose any information about the normal and faulty circuits and is as accurate as any of the other methods. It requires only $\lceil \log_2 n \rceil$ lists compared to the n lists needed for multilist simulation. However, fault list computations depend on signal values and may be more complex than in the multilist method.

2.4 Concurrent simulation

There is no limitation on the number of logic values in this simulation method since faulty and fault-free circuits are treated independently. As long as the primitive elements of the circuit are well defined, the evaluation of faulty circuits presents no difficulty.

2.5 Summary of results

The results of Section II are represented in Table VI.

Deductive simulation with three logic values (indicated by * in Table VI) requires the introduction of the concept of star faults. Deductive fault simulation for more than three logic values (indicated by † in Table VI) could be defined by using a transformed circuit as proposed in Section 2.3. However, the complexity of such a procedure does not seem to justify its use.

From the point of view of simulating more than three logic values, concurrent simulation represents the simplest, most flexible simulation method.

III. MODELING LEVELS

Three levels of modeling and their effects on the simulation method used will be considered: gates, higher-level primitives, and user-defined functions.

3.1 Gate-level simulation

All the fault-simulation methods presented here were initially developed for simulating circuits modeled at the gate level. Therefore, none of the methods presents any problem, provided only two (or three) logic values are to be simulated. The differences due to the number of logic values needed have already been discussed in Section II, and the effects of detailed timing analysis are discussed in Section IV.

3.2 High-level primitives

It is often convenient to model devices such as flip-flops, multiplexors, counters, and shift registers as high-level primitives rather than as interconnections of gates. For purposes of simulation, such devices may be described by tables, Boolean equations, or algorithms. The

Table VI—Summary of Results: logic values

	Parallel	Multilist	Deductive	Concurrent
Switching algebra	One word per line	Two lists per line	Well defined	Well defined
Three logic values	Two words per line	Three lists per line	Well defined but pessimistic*	Well defined
Five logic values	Three words per line	Five lists per line	Undefined†	Well defined
n logic values	$\lceil \log_2 n \rceil$ words per line	n lists per line	Undefined†	Well defined

* Deductive simulation with three logic values.
† Deductive fault simulation for more than three logic values.

type of representation that is most convenient to use will usually depend on the simulation method.

3.2.1 Parallel simulation

Several solutions are possible. The input values for individual faults can be determined from the input word(s), and the outputs of the high-level primitive can be evaluated for each case. The output values must then be packed so that the parallel simulation method may be used elsewhere. While this approach may be satisfactory for predominantly gate-level circuits which also contain a few high-level primitives, the overhead associated with converting to single-fault simulation and back to parallel simulation may not be acceptable.

When only two logic values are involved, the primitive can be represented by Boolean (switching) expressions. The operators in these expressions can be treated exactly like gates in parallel simulation of gate-level circuits. When more than two logic values are simulated, the description of the primitive may be in the form of tables. Using a coding of the type described in Section II, switching algebraic expressions for the coded output words (i.e., the 0-word, 1-word, etc.) can be obtained in terms of the coded words associated with the inputs and state variables. These equations can then be used to compute the coded output words.

3.2.2 Multilist simulation

The function realized by a high-level primitive can be represented by tables. From these tables, equations for the output lists in terms of lists associated with inputs and state variables can be obtained as discussed in Section 2.2 and used for simulation.

3.2.3 Deductive simulation

As in the case of parallel simulation, one approach is to simulate each high-level primitive for one fault at a time and use the results to construct output fault lists for use outside the primitive. Alternatively, outputs and the next state values of state variables may be represented by Boolean equations, which are used for fault-list computations in the same manner as gates. When more than two logic values are to be simulated, a binary coding can be used as discussed in Section 2.3, and equations for each coded bit can be used for fault-list computations.

Another possibility is to use tables that specify the fault-list computations for every combination of input values.[4] These tables can be constructed from the tables specifying the primitive, as shown by the example of Fig. 5.

The behavior of the function is represented in Table VII, where p_1, q_1 are the initial states of the flip-flops and p_2, q_2 are the next states.

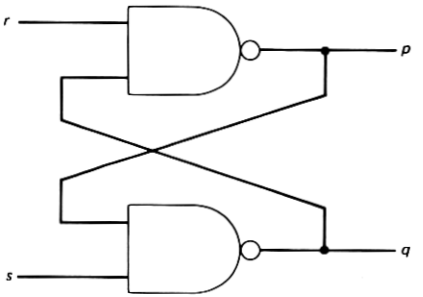

Fig. 5—NAND SR latch.

Table VII—Behavior of SR latch

	p_1	q_1	r	s	p_2	q_2
1	0	1	0	0	1	1
2	0	1	0	1	1	0
3	0	1	1	0	0	1
4	0	1	1	1	0	1
5	1	0	0	0	1	1
6	1	0	0	1	1	0
7	1	0	1	0	0	1
8	1	0	1	1	1	0
9	1	1	0	0	1	1
10	1	1	0	1	1	0
11	1	1	1	0	0	1
12	1	1	1	1	*u*	*u*

The table does not include 00 as an initial state because it cannot be produced directly.

The fault-list propagation is summarized in Table VIII and does not include local faults. P_1, Q_1, R, S, P_2, and Q_2 are the fault lists associated with p_1, q_1, r, s, p_2, and q_2. The star faults in the table are to be added to both the fault lists, P_2 and Q_2. We shall demonstrate the procedure used for deriving Table VIII, by showing how line 1 of the table was obtained.

Consider line 1 of Table VII. To get a change in p_2, we need a change into lines 3, 4, 7, or 11 and for a change in q_2, we need a change into lines 2, 6, 8, or 10, which produces:

$$P_2 = \bar{P}_1\bar{Q}_1R\bar{S} \cup \bar{P}_1\bar{Q}_1RS \cup P_1Q_1R\bar{S} \cup P_1\bar{Q}_1R\bar{S} \cup \bar{P}_1Q_1(\Phi)$$
$$= R\bar{S} \cup \bar{P}_1R$$
$$Q_2 = \bar{P}_1\bar{Q}_1\bar{R}S \cup P_1Q_1RS \cup P_1Q_1\bar{R}S \cup P_1\bar{Q}_1\bar{R}S \cup \bar{P}_1Q_1(\Phi)$$
$$= \bar{R}S \cup Q_1S,$$

where juxtaposition represents intersection. Since a change to line 12 is needed to cause p_2 and q_2 to become unknown, the star faults are

Table VIII—Fault-list equations for SR latch

	P_2	Q_2	Star Faults
1	$R\bar{S}\cup\bar{P}_1R$	$\bar{R}S\cup Q_1S$	$P_1\bar{Q}_1RS$
2	$RS\cup\bar{P}_1R$	$S\cup\bar{P}_1R$	$P_1\bar{Q}_1R\bar{S}$
3	$R\cup Q_1S$	$RS\cup Q_1S$	$P_1\bar{Q}_1\bar{R}S$
4	$R\cup Q_1\bar{S}$	$R\bar{S}\cup Q_1\bar{S}$	$P_1\bar{Q}_1\bar{R}\bar{S}$
5	$R\bar{S}\cup RP_1$	$\bar{R}S\cup\bar{Q}_1S$	$\bar{P}_1Q_1RS$
6	$RS\cup P_1R$	$S\cup P_1R$	$\bar{P}_1Q_1R\bar{S}$
7	$R\cup\bar{Q}_1S$	$RS\cup R\bar{Q}_1$	$\bar{P}_1Q_1\bar{R}S$
8	$\bar{R}S\cup P_1\bar{R}$	$S\cup P\bar{R}$	$\bar{P}_1Q_1\bar{R}\bar{S}$
9	$R\bar{S}\cup Q_1R$	$\bar{R}S\cup SP_1$	$\bar{P}_1\bar{Q}_1RS$
10	$RS\cup Q_1R$	$S\cup Q_1R$	$\bar{P}_1\bar{Q}_1R\bar{S}$
11	$R\cup Q_1S$	$RS\cup Q_1S$	$\bar{P}_1\bar{Q}_1\bar{R}_1S$
12	{}	{}	$\bar{P}_1\bar{Q}_1\bar{R}\bar{S}$

given by

$$P_1\bar{Q}_1RS.$$

We have used $\bar{P}_1Q_1$, which corresponds to the initial state 00 in the faulty circuit, as a don't-care state (Φ) to simplify the expressions.

3.2.4 Concurrent simulation

In concurrent simulation, the same method is used to evaluate fault-free and faulty circuit signal values. Therefore, no transformations of representation are necessary, and any representation that leads to efficient simulation may be chosen.

3.3 User-defined functions

Our discussion of Section 3.2 also applies to user-defined functions. The main difference is that the tables or equations used for representing the functions must be generated from descriptions in a high-level language such as the function definition language in LAMP.[20]

A typical construct in such a language is the cause-effect statement. Such statements can be nested to many levels. The techniques discussed in Section 3.2 can be used for simulating user-defined functions using the parallel, multilist, or deductive method by first replacing cause-effect statements by equivalent equations. For example, the statement

$$\text{if } x \quad \text{then} \quad z = a \quad \text{else} \quad z = b$$

can be replaced by

$$z = a\cdot x + b\cdot\bar{x} + a\cdot b.$$

The redundant term $a\cdot b$ has been introduced to produce the correct result $z = 1$ for the case $a = b = 1$ and $x = u$.[21] Otherwise, the pessimistic result $z = u$ will be produced for this case.

Concurrent simulation does not require the transformation of cause-

effect statements into equations. For each fault and each combination of inputs and state, only those computations enabled by the conditions need be performed. The operations in a function definition need not be restricted to logical operations. Therefore, it is not necessary to generate Boolean equations corresponding to arithmetic operations, as would be necessary in the other methods considered. Thus, it appears that the concurrent method would allow simulation of functions defined at a higher level than is possible with the other methods.

3.4 Summary of results

The results of this section are summarized in Table IX. The concurrent method is clearly superior in its ability to simulate different levels of models.

IV. TIMING

In this section, we study different effects related to timing, and their impact on the four simulation methods under consideration. We shall consider the effects of different rise and fall times associated with signal changes, suppression of short pulses to model inertial delays, and the simulation of faults which affect the magnitude of delays associated with devices. We shall restrict our discussion to logic simulation with two and three logic values.

4.1 Rise and fall times

The delays associated with 0 to 1 and 1 to 0 transitions of a signal, called here the rise and fall times, are not necessarily equal.[22,23] All the methods of fault simulation under discussion simulate a number of signals simultaneously, some of which may be rising and some falling. To simulate this effect accurately, a mechanism is necessary for allowing rising and falling signals to change at different times.

Let t_0 be a time before any change occurs on the line under consideration. Due to differences in the rise and fall times, signal changes may occur on the line at times t_1 and t_2, where $t_0 < t_1 \leq t_2$. Thus, at time t_2, all signal changes associated with the particular event would have occurred. The effect of different rise and fall times can be

Table IX—Summary of results: modeling levels

	Parallel	Multilist	Deductive	Concurrent
Gate level	1	1	1	1
Higher level primitives	2	2	2	1
User defined functions	2	2	2	1

Note: 1 = No transformations required.
2 = Transformation into equations required.

simulated accurately by computing the values of the signals at time t_1 based on the values at t_0 and t_2, namely, the initial and final values for the particular set of transitions.

In the following sections, we shall consider four simulation methods and examine the results produced by their different models at three points in time, namely, before $t_1(t_0)$, between t_1 and $t_2(t_1)$, and after $t_2(t_2)$.

4.1.1 Parallel simulation

Let ξ_0, ξ_1, and ξ_2 be the words associated with a line x at times t_0, t_1, and t_2, in two-valued parallel simulation. In three-valued simulation, two words denoted by superscripts 0 and 1 will be associated with the line for each of the above times, and the coding of Table III will be used.

Case 1: Rise time < fall time. We have

$$\xi_1 = \xi_0 + \xi_2$$

for two-valued simulation, and

$$\xi_1^1 = \xi_0^1 + \xi_2^1$$
$$\xi_1^0 = \xi_0^0 \cdot \xi_2^0$$

for three-valued simulation where + and · represent bitwise OR and AND performed on full words.

Case 2: Fall time < rise time. We have

$$\xi_1 = \xi_0 \cdot \xi_2$$

for two-valued simulation, and

$$\xi_1^1 = \xi_0^1 \cdot \xi_2^1$$
$$\xi_1^0 = \xi_0^0 + \xi_2^0$$

for three-valued simulation.

The preceding formulas can be verified by checking all nine possible transitions between the set $\{0, 1, u\}$ and itself.

4.1.2 Three-list method

Let X_0^i, X_1^i, and X_2^i be the i-lists at times t_0, t_1, and t_2, defined above, for $i = 0, 1, u$. Using the same arguments as in Section 4.1.1, we obtain the lists for time t_1 as given below.

Case 1: Rise time < fall time. We have

$$X_1^1 = X_0^1 \cup X_2^1$$
$$X_1^0 = X_1^0 \cap X_2^0.$$

Case 2: Fall time < rise time. We have

$$X_1^1 = X_0^1 \cap X_2^1$$

$$X_1^0 = X_0^0 \cup X_2^0.$$

In both cases, $X_1^u = (\overline{X_1^0 \cup X_1^1})$.

4.1.3 Deductive simulation

Deductive simulation with different rise and fall times has been discussed by Kjelkerud and Thessen.[24] Here we present an alternate method.

Let the times t_0, t_1, and t_2 be as defined earlier and let x_i and X_i represent the signal values and fault lists at those times, $i = 0, 1, 2$. If the rise time is less than the fall time, all 0 to 1 transitions will occur at t_1. Therefore, we have

$$x_1 = x_0 + x_2.$$

Similarly, if fall time < rise time, 1 to 0 transitions will occur at t_1, and $x_1 = 1$ if and only if it remains at 1 throughout the transitions. Therefore, for this case

$$x_1 = x_0 \cdot x_2.$$

The fault lists X_1 at time t_1 for different signal changes in the fault-free circuit and different relative values of rise and fall times can be determined from these equations. They are summarized in Table X.

4.1.4 Concurrent simulation

In fact, concurrent simulation is a trivial case, because fault-free and and faulty circuits are simulated independently. Rising and falling edges will still occur in distinct event waves, but the treatment of these events is individual.

4.2 High-frequency rejection

High-frequency rejection consists of eliminating short pulses for modeling the effect of inertial delays. We consider events occurring at

Table X—Fault-list equations for handling different rise and fall times

Fault-Free Circuit	Rise Time < Fall Time	Fall Time < Rise Time
Rising edge $x_0 = 0; x_2 = 1$	$x_1 = 1$ $X_1 = \overline{X}_0 \cap X_2$	$x_1 = 0$ $X_1 = X_0 \cap \overline{X}_2$
Falling edge $x_0 = 1; x_2 = 0$	$x_1 = 1$ $X_1 = X_0 \cap \overline{X}_2$	$x_1 = 0$ $X_1 = \overline{X}_0 \cap X_2$
Constant one $x_0 = 1; x_2 = 1$	$x_1 = 1$ $X_1 = X_0 \cap X_2$	$x_1 = 1$ $X_1 = X_0 \cup X_2$
Constant zero $x_0 = 0; x_2 = 0$	$x_1 = 0$ $X_1 = X_0 \cup X_2$	$x_1 = 0$ $X_1 = X_0 \cap X_2$

times t_0, t_1, and t_2, where $t_0 < t_1 \leq t_2$ and the logic values at these times. If $t_2 - t_1$ is less than the magnitude of the inertial delay, then the change at t_1 must be rejected to suppress short pulses and the value between t_1 and t_2, x_1, will be replaced by a corrected logic value, x_{1n}. If x_0, x_1, and x_2 are the computed signal values at t_0, t_1, and t_2 respectively, and if $t_2 - t_1$ is less than the inertial delay, the corrected signal value at time t_1 is given by:

$$x_{1n} = x_0x_1 + x_1x_2 + x_0x_2.$$

The method for performing high-frequency rejection can be derived from this equation.

In case there are more than two events within the range of the inertial delay, the treatment elaborated above must be repeated for each pair of events within that range. For example, consider three events occurring at times t_1, t_2, and t_3. The following triples will be considered: (t_0, t_1, t_2), (t_0, t_1, t_3), (t_1, t_2, t_3), which represent three pairs of events.

4.2.1 Parallel simulation

For two-valued parallel simulation, the word ξ_1 has to be replaced by

$$\xi_{1n} = \xi_0\xi_1 + \xi_1\xi_2 + \xi_0\xi_2$$

and for three-valued parallel simulation, ξ_1^0 and ξ_1^1 are replaced by

$$\xi_{1n}^1 = \xi_0^1\xi_1^1 + \xi_1^1\xi_2^1 + \xi_0^1\xi_2^1$$

$$\xi_{1n}^0 = \xi_0^0\xi_1^0 + \xi_1^0\xi_2^0 + \xi_0^0\xi_2^0.$$

The coding defined in Table III was used to obtain the above equations for three-valued parallel simulation.

4.2.2 Three-list methods

Using the equation for x_{1n} given above, we obtain the following fault-list equations:

$$X_{1n}^0 = (X_0^0 \cap X_1^0) \cup (X_1^0 \cap X_2^0) \cup (X_0^0 \cap X_2^0)$$

$$X_{1n}^1 = (X_0^1 \cap X_1^1) \cup (X_1^1 \cap X_2^1) \cup (X_0^1 \cap X_2^1)$$

$$X_{1n}^u = (\overline{X_1^0 \cup X_1^1}).$$

4.2.3 Deductive simulation

The deductive fault list X_{1n} can be obtained from the equation for the new signal value x_{1n}, the values of x_0, x_1, and x_2, and the associated fault lists. The fault list X_{1n} can be computed in the same manner as fault propagation through functional blocks.[19] In fact, high-frequency

rejection may be thought of as being performed by a filter whose equation is given above.

The fault-list computations for the eight possible patterns of x_0, x_1, and x_2 are summarized in Table XI.

As an example, consider the case $x_0 = 0$, $x_1 = 1$, $x_2 = 1$(line 4 in Table XI). We have

$$x_{1n} = x_0x_1 + x_1x_2 + x_0x_2 = a + b + c.$$

The fault lists associated with the terms $a = x_0x_1$, $b = x_1x_2$, and $c = x_0x_2$ for the specified values are:

$$A = X_0 \cap \bar{X}_1; \quad B = X_1 \cup X_2; \quad C = X_0 \cap \bar{X}_2.$$

Therefore,

$$\begin{aligned} X_{1n} = \bar{A} \cap B \cap \bar{C} &= (\bar{X}_0 \cup X_1) \cap (X_1 \cup X_2) \cap (\bar{X}_0 \cup X_2) \\ &= (\bar{X}_0 \cap X_1) \cup (\bar{X}_0 \cap X_2) \cup (X_1 \cap X_2). \end{aligned}$$

4.2.4 Concurrent simulation

In this case, each faulty signal value is computed separately. Therefore, high-frequency rejection can be performed on each signal individually, using the equation for x_{1n} given in the preceding section.

4.2.5 Suppression of short-duration detections

We have considered the suppression of short pulses produced independently by each fault-free or faulty signal value. However, we did not consider the case of a short pulse of detection, when neither the faulty nor the fault-free signal incurs a pulse. This is illustrated by the case where $x_0 = 1$, $x_1 = 1$, $x_2 = 0$, for the fault-free signal and $x_0 = 1$, $x_1 = 0$, $x_2 = 0$, for the faulty signal. This causes a short detection between t_1 and t_2. For deductive simulation, this short detection may be eliminated by using the formula

$$X_{1n} = (X_0 \cap X_1) \cup (X_0 \cap X_2) \cup (X_1 \cap X_2)$$

independently of the fault-free signal pattern and after the high-

Table XI—Fault-list equations for high-frequency rejection

x_0	x_1	x_2	x_{1n}	X_{1n}
0	0	0	0	$(X_0 \cap X_1) \cup (X_1 \cap \underline{X}_2) \cup (X_0 \cap \underline{X}_2)$
0	0	1	0	$(X_0 \cap \underline{X}_1) \cup (\underline{X}_1 \cap \overline{X}_2) \cup (X_0 \cap \overline{X}_2)$
0	1	0	0	$(\underline{X}_0 \cap \overline{X}_1) \cup (\overline{X}_1 \cap X_2) \cup (\underline{X}_0 \cap X_2)$
0	1	1	1	$(\underline{X}_0 \cap X_1) \cup (X_1 \cap X_2) \cup (\underline{X}_0 \cap X_2)$
1	0	0	0	$(\overline{X}_0 \cap \underline{X}_1) \cup (\underline{X}_1 \cap X_2) \cup (\overline{X}_0 \cap X_2)$
1	0	1	1	$(X_0 \cap \overline{X}_1) \cup (\overline{X}_1 \cap \underline{X}_2) \cup (X_0 \cap \underline{X}_2)$
1	1	0	1	$(X_0 \cap X_1) \cup (X_1 \cap \overline{X}_2) \cup (X_0 \cap \overline{X}_2)$
1	1	1	1	$(X_0 \cap X_1) \cup (X_1 \cap X_2) \cup (X_0 \cap X_2)$

frequency rejection has been performed. The term X_i is the set of faults detected at time t_i.

The same method may be used for all the other simulation algorithms described earlier.

4.3 Delay faults

A fault that affects the transport delay associated with a signal is called a delay fault. Consider a fault that causes a delay to change from d to d'. When the signal at the site of such a fault changes, the signal value corresponding to the particular faulty circuit must be delayed by d' instead of d.

4.3.1 Parallel simulation

Two aspects of delay faults must be considered: injection of delay faults and the propagation of the effects of delay faults. Let us assume that a gate which is the site of a delay fault has been evaluated at time t, and the jth bit of the word represents the circuit with the delay fault. Let the normal and faulty delays be d and d', respectively, and let $d < d'$. Let $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ and $\mathbf{x}' = (x_1', x_2', \cdots, x_n')$ be the vectors representing the old and new values, respectively. If for any $i \neq j$, $x_i' \neq x_i$, a vector $(x_1', x_2', \cdots, x_{j-1}', x_j, x_{j+1}', \cdots, x_n')$ will be scheduled to be applied to the gate output at time $t + d$. If $x_j' \neq x_j$, the vector $\mathbf{x}'$ will be scheduled to be applied to the gate output at time $t + d'$. Similarly, if $d' < d$ and $x_j' \neq x_j$, the vector $(x_1, x_2, \cdots, x_{j-1}, x_j', x_{j+1}, \cdots, x_n)$ will be scheduled for time $t + d'$. If $x_i' \neq x_i$ for any $i \neq j$, then the vector $\mathbf{x}'$ will be scheduled to be applied at time $t + d$. The vectors for updating at the different times can be obtained from the old and new vectors by appropriate masks and logical operations.

From the above discussion it should be clear that the effect of delay faults is to cause the signal values on the same lead in the presence of delay faults to change at different times. When one or more values in a vector change, the gates to which the signal fans out in the fault-free and all faulty circuits are scheduled for evaluation in parallel. Therefore, no special treatment is necessary for propagating delay faults. Since any signal change at the inputs of a device, faulty or fault-free, will cause an evaluation of the device (faulty and fault-free), delay faults will tend to increase the number of evaluations required.

4.3.2 Three-list method

The equations required for simulating delay faults using the three-list and deductive methods can be derived by treating the delay fault as an internal fault in a functional block. These equations can then be used for simulating delay faults without explicitly modeling them as faults in functions.

Let α be a delay fault which causes the delay associated with a signal to change from d to d'. Let f_α be a fault variable,[19] which has the fault-free value of 0, but the fault α causes it to become 1. We shall represent the input and the output of the function used for modeling the delay fault by x and z, respectively. We assume that the evaluation is being done at time $t = 0$, and the value of the input x, t_1 units of time earlier is represented by $x(-t_1)$. Two different cases must be considered:

Case 1: $d < d'$. The value of z at time d is given by

$$\text{if } f_\alpha \text{ then } z = x(d - d')$$
$$\text{else } z = x,$$

which can be transformed into the equation

$$z(d) = f_\alpha \cdot x(d - d') + \bar{f}_\alpha \cdot x.$$

Case 2: $d' < d$. The value of z at time d' can be represented by a function as in Case 1, and transformed into the following equation:

$$z(d') = f_\alpha \cdot x + \bar{f}_\alpha \cdot x(d' - d).$$

The equations for the three-list method can be obtained from these equations using the method discussed in Section 2.2.

Case 1: $d < d'$

$$Z^1(d) = [\{\alpha\} \cap X^1(d - d')] \cup [X^1 \cap \overline{\{\alpha\}}]$$

$$Z^0(d) = [\overline{\{\alpha\}} \cap X^0] \cup [X^0 \cap X^0(d - d')] \cup [X^0(d - d') \cap \{\alpha\}]$$

$$Z^u(d) = [\overline{Z^1(d) \cup Z^0(d)}].$$

Case 2: $d' < d$

$$Z^1(d') = [\{\alpha\} \cap X^1] \cup [X^1(d' - d) \cap \overline{\{\alpha\}}]$$

$$Z^0(d') = [\overline{\{\alpha\}} \cap X^0] \cup [X^0 \cap X^0(d' - d)] \cup [X^0(d' - d) \cap \{\alpha\}]$$

$$Z^u(d') = [\overline{Z^1(d') \cup Z^0(d')}].$$

4.3.3 Deductive simulation

The functional equations derived in Section 4.3.2 can be used for deriving deductive fault lists for delay faults. The fault-list equations will depend on signal values as shown in Table XII.

4.3.4 Concurrent simulation

Since each fault is handled separately, the simulation of delay faults does not need any special processing.

Table XII—Fault-list equations for delay faults

x	$x(d-d')$	$z(d)$	$Z(d)$
0	0	0	$[X(d-d') \cap \{\alpha\}] \cup \left[X \cap \overrightarrow{\{\alpha\}}\right]$
0	1	0	$[\overline{X(d-d')} \cap \{\alpha\}] \cup \left[X \cap \overrightarrow{\{\alpha\}}\right]$
1	0	1	$\left[X \cap \overrightarrow{\{\alpha\}}\right] \cup [X(d-d') \cap \{\alpha\}] \cup [X \cap \overline{X(d-d')}]$
1	1	1	$[X(d-d') \cap \{\alpha\}] \cup [X \cap X(d'-d)] \cup \left[X \cap \overrightarrow{\{\alpha\}}\right]$

Case 1: $d < d'$

x	$x(d'-d)$	$z(d')$	$Z(d')$
0	0	0	$[X \cap \{\alpha\}] \cup \left[X(d'-d) \cap \overrightarrow{\{\alpha\}}\right]$
0	1	1	$[\overline{X} \cap X(d'-d)] \cup [\overline{X} \cap \{\alpha\}] \cup \left[X(d'-d) \cap \overrightarrow{\{\alpha\}}\right]$
1	0	0	$[\overline{X} \cap \{\alpha\}] \cup \left[X(d'-d) \cap \overrightarrow{\{\alpha\}}\right]$
1	1	1	$\left[X(d'-d) \cap \overrightarrow{\{\alpha\}}\right] \cup [X \cap \{\alpha\}] \cup [X \cap X(d'-d)]$

Case 2: $d' < d$

4.4 Summary of results

Changes in the fault-free and faulty values may occur at different times for the same line due to different rise and fall times and to delay faults. In the case of parallel simulation, a change in a single faulty or fault-free value on a line leads to computations involving the whole word (or pair of words). In the three-list and deductive methods, the addition or deletion of a single fault will require recomputation of complete lists. On the other hand, concurrent simulation treats each event, faulty or fault-free, independent of all other events and, therefore, should require less computation. High-frequency rejection is also simpler in concurrent simulation than in the other methods.

V. CONCLUSION

We have compared parallel, multilist, deductive, and concurrent simulation methods with regard to their ability to simulate more than two logic values, different levels of simulation, and accurate timing

analysis. All the methods, except deductive, can handle any number of logic values without significant changes in the method. An extension of the deductive method to an arbitrary number of logic values is presented. Concurrent simulation appears to be the most convenient method of simulating an arbitrary number of logic values.

All the methods, except concurrent, require the transformation of functional descriptions of high-level devices into Boolean equations. No such transformation is required for concurrent simulation. In fact, it is not even necessary to restrict operations in functional descriptions to Boolean operations if concurrent simulation is used.

All the methods are capable of handling different rise and fall times, performing high-frequency rejection and simulating delay faults. Since concurrent simulation handles each event separately, these functions can be performed more easily and efficiently than the other methods.

In addition to the aspects discussed here, two factors that must be considered in selecting a simulation method are storage requirements and speed. A detailed analysis of the speed and the storage requirements of these methods is made in Ref. 11 based upon statistical data gathered from deductive simulation.

REFERENCES

1. S. Seshu, "The Logic Analyzer and Diagnosis Programs," Coordinated Science Laboratory, Rept. R-226, 1964.
2. S. Seshu, "On an Improved Diagnosis Program," IEEE Trans. Electronic Computers, *EC-14*, No. 1 (February 1965), pp. 76–9.
3. S. A. Szygenda, "TEGAS-2- Anatomy of a General Purpose Test Generation and Simulation System for Digital Logic," Proc. 9th ACM-IEEE Design Automation Workshop (June 1972), pp. 116–27.
4. D. B. Armstrong, "A Deductive Method of Simulating Faults in Logic Circuits," IEEE Trans. Computers, *C-21*, No. 5 (May 1972), pp. 464–71.
5. E. G. Ulrich and T. G. Baker, "Concurrent Simulation of Nearly Identical Digital Networks," Computer, *7*, No. 4 (April 1974), pp. 39–44.
6. H. Y. Chang et al., "Comparison of Parallel and Deductive Simulation Methods," IEEE Trans. Computers, *C-23*, No. 11 (November 1974), pp. 1132–8.
7. Y. H. Levendel and W. C. Schwartz, "Impact of LSI on Logic Simulation," Proc. of COMPCON, San Francisco, February 1978.
8. M. Abramovici, M. A. Breuer, and K. Kumar, "Concurrent Fault Simulation and Functional Level Modeling," Proc. 14th Design Automation Conference (June 1977), pp. 128–37.
9. F. Ozguner, W. E. Donath, and C. W. Cha, "On Fault Simulation Techniques," J. Design Automation and Fault Tolerant Computing," *3*, No. 2 (April 1979), pp. 83–92.
10. Y. H. Levendel and P. R. Menon, "Comparison of Fault Simulation Methods — Treatment of Unknown Signal Values," J. of Digital Systems, *4*, No. 4 (Winter 1980), pp. 443–59.
11. Y. H. Levendel and P. R. Menon, unpublished work.
12. Y. H. Levendel and P. R. Menon, "Unknown Signal Values in Fault Simulation," Proc. 9th International Symposium on Fault Tolerant Computing (June 1979), pp. 125–8.
13. M. Yoeli and S. Rinon, "Application of Ternary Algebra to the Study of Static Hazards," *J. ACM, 11*, No. 1 (January 1964), pp. 84–97.
14. E. B. Eichelberger, "Hazard Detection in Combinational and Sequential Switching Circuits," Proc. 5th Annual Symp. on Switching Circuit Theory and Logical Design (1964), pp. 111–20.

15. R. L. Wadsack, "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits," B.S.T.J. *57,* No. 5 (May–June 1978), pp. 1449–74.
16. Y. H. Levendel, P. R. Menon, and C. E. Miller, "Accurate Simulation Models for TTL Totempole and MOS Gates and Tristate Devices," B.S.T.J., *60,* No. 7 (September 1981), pp. 1271–87.
17. Y. H. Levendel and M. A. Breuer, "Vector Representation of Switching and Three-Valued Functions," Proc. Eighth Internat. Symp. on Multi-valued Logic (May 1978), pp. 163–70.
18. S. G. Chappell, C. H. Elmendorf, and L. D. Schmidt, "LAMP: Logic-Circuit Simulators," B.S.T.J., *53,* No. 8 (October 1974), pp. 1451–76.
19. P. R. Menon and S. G. Chappell, "Deductive Fault Simulation with Functional Blocks," IEEE Trans. Computers, *C-27,* No. 8 (August 1978), pp. 689–95.
20. S. G. Chappell et al., "Functional Simulation in the LAMP System," J. Design Automation and Fault Tolerant Computing, *1,* No. 3 (May 1977), pp. 203–16.
21. K. Wu, Ph.D. Dissertation, "Synthesis of Accurate and Efficient Functional Modeling Techniques for Performing Design Verification of VLSI Digital Circuits," Univ. of Texas, Austin, December, 1979.
22. S. G. Chappell and S. S. Yau, "Simulation of Large Asynchronous Logic Circuits Using an Ambiguous Gate Model," Proc. Fall Joint Computer Conf. (1971), pp. 651–61.
23. S. A. Syzgenda, D. M. Rouse, and E. W. Thompson, "A Model and Implementation of a Universal Time Delay Simulator for Large Digital Nets," Proc. Spring Joint Computer Conf. (1970), pp. 207–16.
24. E. Kjelkerud and O. Thessen, "Techniques for Generalized Deductive Fault Simulation," J. Design Automation and Fault Tolerant Computing, *1,* No. 10 (October 1974), pp. 377–90.