Fault Simulation of MOS Digital Circuits

Randal E. Bryant and Michael D. Schuster
Department of Computer Science,
California Institute of Technology
Pasadena, CA

Test engineers use fault simulators to determine how well a sequence of test patterns applied to the inputs of an integrated circuit can distinguish a good chip from a defective one. The fault simulator is given a description of the good circuit, a set of hypothetical faults in the circuit, a specification of the observation points of the test (*e.g.*, the output pins of the chip), and a sequence of test patterns. It then simulates how the good circuit and all of the faulty circuits would behave when the test patterns are applied to the inputs. A fault is considered detected if at any time the simulation of that particular faulty circuit produces a different logic value at some observation point than the simulation of the good circuit produces. By keeping track of which faults have been detected and which have not, the fault simulator can determine the *fault coverage* of the test sequence, which is defined as the ratio of the number of faults detected to the total number simulated. The simulator can also give the user information about which faults have not been detected, either because the test sequence failed to exercise the defective part of the circuit, or because the sequence failed to make the effect of such an exercise visible at an observation point. This information guides the engineer in extending or modifying the test sequence to improve its fault coverage. Such a tool is invaluable in developing test patterns for today's complex digital systems.

For a large integrated circuit such as a microprocessor chip, many faults (*e.g.*, more than 1000) must be simulated for adequate characterization of the fault coverage of a test sequence. Furthermore, the test sequences can involve thousands of patterns. Hence, a simple *serial* simulation (in which the good circuit and each faulty circuit are simulated separately) would require far too much computation. Fortunately, certain clever algorithms can reduce the amount of computation considerably. A technique known as *concurrent simulation* (Ulrich and Baker 1974) exploits the fact that each faulty circuit typically differs only slightly from a good circuit. Instead of simulating each circuit separately, it simulates only the good chip in its entirety. For each faulty circuit, this technique keeps track of how the network state of that circuit differs from the network state of the good circuit by selectively simulating portions of the faulty network. To the user, it appears as if the program is simulating many circuits concurrently, but the amount of CPU time required is only slightly (*e.g.*, less than 10 times) greater than the time required to simulate the good circuit alone. Furthermore, the simulator can easily determine when at an observation point a faulty circuit produces a value that differs from the value produced by the good circuit, without storing the entire output history of the good circuit simulation. Once a fault has been detected, the simulation of this particular circuit can be dropped, thereby reducing the amount of computational required for the rest of the simulation. The faults that cause great differences from the behavior of the good circuit, and that therefore require the most computation effort, also are detected quickly. Therefore, this fault-dropping technique greatly improves the overall performance of the simulator.

Most existing logic simulators model a digital circuit as a network of logic gates, in which each gate produces values on its outputs based on the values applied to its inputs, and possibly based on the value of its internal state. Some of these simulators extend the simple Boolean gate model (in which only the value 0 or 1 is permitted on each input and output) by using additional logic values and special types of gates to model circuit structures such as buses and pass transistors. These simulators are not suitable for modeling faults in MOS digital circuits for the following two reasons: First, many MOS circuit structures cannot be adequately modeled as a set of logic gates. Creating gate-level descriptions of pass-transistor networks, precharged logic, and dynamic memory elements is at best tedious and inaccurate, and at worst impossible—even with extended gate models. The user must inevitably translate the logic design by hand into a form compatible with the simulator. Second, logic-gate simulators are especially poor at predicting the behavior of a MOS circuit in the presence of faults. Even simple logic gates can become seemingly complex sequential circuits when a fault such as an open-circuited transistor occurs (Wadsack 1978; Galiay *et al.* 1980). As a result, fault simulators based on logic gates can model only a limited class of faults, such as gate outputs and inputs stuck at zero or stuck at one. Faults such as short circuits across transistors and between wires, or open circuits in transistors or wires, are beyond their capability.

To remedy these problems with logic-gate simulators, we propose that fault simulations of MOS circuits be performed at the *switch level* with the transistor structure of the circuit represented explicitly, but with each transistor modeled in a very idealized way. This approach has proved

gate state	n-type	p-type	d-type
0	0	1	1
1	1	0	1
X	X	X	1

TABLE 1. Transistor state as a function of its type and as a function of the state of its gate node.

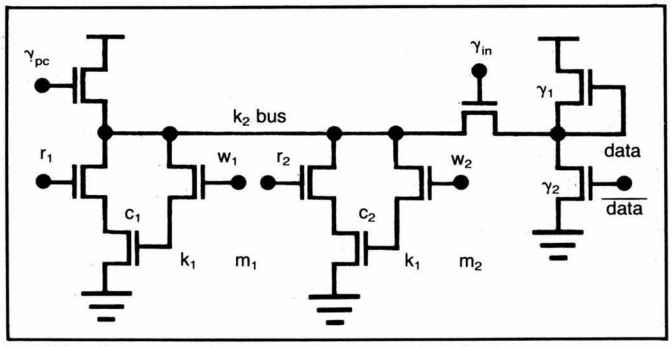


FIGURE 1. Three-transistor dynamic RAM.

successful for logic simulation in programs such as MOSSIM (Bryant 1980 and 1981) and MOSSIM II (Bryant *et al.* 1982), because properties such as the bidirectional nature of field-effect transistors and the charge storage capabilities of the nodes in a MOS circuit are modeled directly, instead of through an artificial translation into logic gates.

We have adapted the technique of concurrent simulation to implement a fault simulator for MOS circuits, for which the problem is viewed as one of simulating many nearly identical switch-level networks. This program FMOSSIM can simulate many MOS circuits under a variety of fault conditions at much higher speeds than would be possible with serial simulation. Other concurrent fault simulators for MOS have been implemented (Bose *et al.* 1982), but these can model only a very limited class of networks. In this article we present an overview of the switch-level model, and explain how different faults can be represented in it. We also present some performance results from FMOSSIM.

The Network Model

The following network model is implemented in the simulators MOSSIM II and FMOSSIM. It includes a more general transistor model than other switch-level simulators do, giving better capabilities for fault injection. A switch-level network consists of a set of *nodes* connected by a set of *transistors*. Each node has a state 0, 1 or X, where 0 and 1 represent low and high voltages, respectively. The X state represents an indeterminate voltage arising from an uninitialized node, from a short circuit, or from improper charge-sharing. No restrictions are placed on how transistors are interconnected.

Each node is classified as either an *input* node or a *storage* node. An input node provides a strong signal to the network, as does a voltage source in an electrical circuit. Its state is not affected by the actions of the network. Examples include the power and ground nodes V_{DD} and **GND**, which act as constant sources of 1 and 0 voltages, respectively, as well as any clock or data inputs. The state of a storage node is determined by the operation of the network. Much like a capacitor in an electrical circuit, a storage node holds its state in the absence of connections to input nodes. To provide a simple model of charge sharing, each node is assigned a discrete *size* from the set $\{k_1,k_2,\ldots,k_q\}$ where the value on a larger node overrides the value on a smaller one when the nodes share charge. The number of different sizes q required depends on the circuit to be simulated. Most circuits can be represented with just two node sizes. In this representation, high-capacitance nodes such as buses are assigned size k_2, and all other nodes are assigned size k_1.

A transistor is a device with terminals labeled *gate*, *source*, and *drain*. No distinction is made between the source and drain connections; each transistor is symmetric and bidirectional. Because transistors can be either *n-type*, *p-type*, or *d-type*, both nMOS and CMOS circuits can be modeled. A d-type transistor corresponds to a negative-threshold depletion-mode device. A transistor acts as a resistive switch connecting or disconnecting its source and drain nodes according to its type and the state of its gate node, as shown in Table 1. Transistor states 0 and 1 represent open (nonconducting) and closed (fully conducting) conditions, respectively. The X gate state represents an indeterminate condition between open and closed, inclusive.

To model the behavior of ratioed circuits, each transistor is assigned a discrete *strength* from the set $\{\gamma_1, \gamma_2,\ldots, \gamma_p\}$, in which a stronger transistor is assumed to have much greater conductance than a weaker one. The total number of strengths p required depends on the circuit to be modeled. Most CMOS circuits do not use ratioed logic, and hence can be modeled with just one transistor strength. Most nMOS circuits require only two strengths. Pull-up loads are assigned strength γ_1, and all others are assigned strength γ_2. (In some cases, more strengths are required.)

As an example of a switch-level network, consider the three-transistor dynamic RAM circuit shown in Figure 1. The bus node has size k_2, to indicate that it can supply its state to the size k_1 storage node (either m_1 or m_2) of the selected memory element during a WRITE operation (when w_1 or w_2 is 1) and to the size k_1 drain node (either c_1 *or* c_2) of the storage transistor during a READ operation (when r_1 or r_2 is 1). The d-type *pull-up* transistor in the input inverter has strength γ_1, to indicate that it can drive the bus high only when the *pull-down* transistor having strength γ_2 is not conducting. The strengths of all other transistors in the circuit are arbitrary, because they are not involved in ratioed path formation (except possibly when faults are present).

The behavior of a switch-level network is described by its *steady-state response*. This parameter can be defined informally as the set of states that would form on the storage nodes for a particular set of transistor states, input node states, and initial storage node states, assuming the transistors are held fixed. This response is computed in both MOSSIM II and FMOSSIM by the solution of a set of equations in a simple discrete algebra (Bryant 1983). For

each set of inputs, a circuit is simulated with a series of *unit step functions* until a stable state is reached. Each unit step involves computing the steady-state response of the network, setting the storage nodes to these values, and setting the transistors according to the states of their gate nodes. This simulation can proceed very quickly, because it re-computes the node states only for those parts of the network where activity occurs.

Fault Injection

Conceptually, faults are represented in FMOSSIM as though extra *fault transistors* were added to the network. In the implementation, however, many of these faults are injected without actually adding the fault transistors; nevertheless, the behavior is equivalent to that described below. The gate nodes of the fault transistors are considered to be extra *fault inputs* to the network that control the presence or absence of the failures. A variety of MOS failures can be modeled with this method. For example, a short circuit between two nodes is modeled by connecting the nodes with a fault transistor that is open in the good circuit and closed in the faulty circuit. Similarly, an open circuit is modeled by splitting a node into two parts and connecting the resulting nodes with a fault transistor that is closed in the good circuit and open in the faulty circuit. By adjusting the strength of the fault transistor, the resistance of the short or open can be modeled in an approximate way. For example, if the strength of the fault transistor is set to γ_{p+1} (*i.e.*, a strength greater than that of any normal transistor), then setting this transistor state to 1 shorts the source and drain nodes together such that they act as a single node. Moreover, because the state of each fault transistor can be controlled independently, both single and multiple faults can be injected.

Figure 2 illustrates the use of fault transistors to create a variety of circuit faults. The transistors with gate nodes labled f are normally 0, but are set to 1 to create the fault; the transistors with gate nodes labeled $\overline{f}$ are normally 1, but are set to 0 to create the fault. A stuck-at-zero or stuck-at-one node fault can be modeled by inserting a strength γ_{p+1} fault transistor to short the node to GND or to V_{DD}, respectively. A stuck-closed transistor fault is injected by shorting the transistor's source and drain together with a fault transistor whose strength equals that of the failing transistor. Similarly, a stuck-open transistor fault is modeled by putting a fault transistor in series with it. In FMOSSIM, both stuck-at node states and stuck-at transistor states are implemented without extra fault transistors, while other faults require that additional transistors be inserted in the network.

Performance Results

As a test case for evaluating the performance of FMOSSIM, we simulated a 64-bit dynamic RAM circuit containing 374 transistors. This circuit incorporates a variety of MOS structures such as logic gates, bidirectional pass transistors, dynamic latches, precharged buses, and three-transistor dynamic memory elements. The circuit was simulated with 428 faults—each storage node stuck at 0, each storage node stuck at 1, and pairs of adjacent buses shorted together. To validate the program, we also simulated other faults, including transistors stuck open and closed. The simulator was implemented in the MAINSAIL programming language (Xidak 1982), and executed on a DEC-20/60.

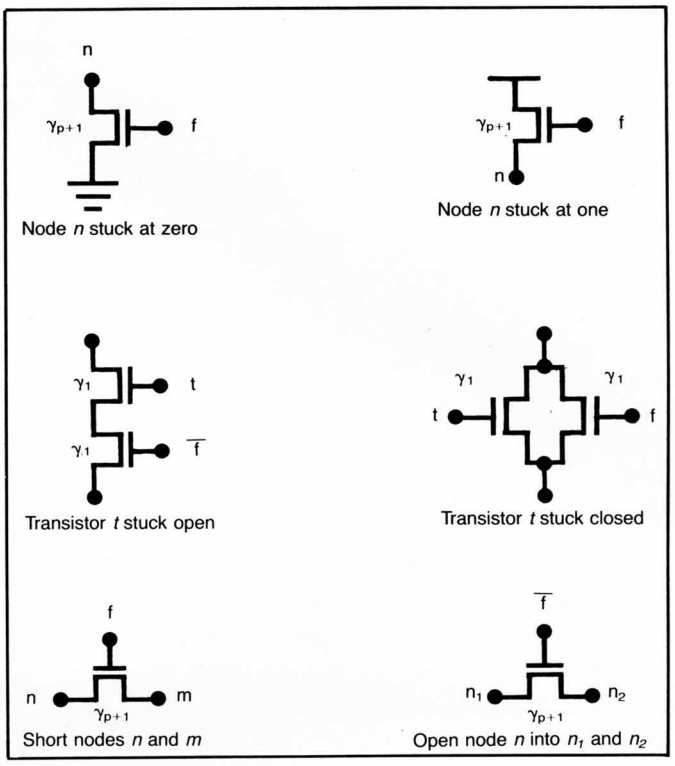


FIGURE 2. Modeling MOS failures with fault transistors.

Figure 3 illustrates the performance of FMOSSIM in simulating a test sequence consisting of a marching test (Winegarden and Pannell 1981) of the memory, together with special tests for the control logic. The curve climbing diagonally upward indicates the total number of faults detected as the test progresses. All faults were detected after 407 patterns. The falling curve indicates the CPU time required to simulate each pattern. This time started at 27 seconds when the circuits were initialized. However, after 100 patterns, it dropped to around 1 second as faults were detected and the simulations of these circuits were dropped. This time finally reached 0.3 seconds at the end, when only the good circuit was being simulated.

Figure 4 illustrates the performance advantage of concurrent simulation over simulating each faulty circuit separately. The curve falling diagonally to the right indicates the number of circuits being simulated as the test proceeds. The other curve indicates the CPU time required to simulate each pattern divided by the number of circuits being simulated for that pattern. This curve started at about 0.05 seconds per pattern, dropped to a low of 0.005 seconds once those faults causing major differences from the good circuit were dropped, and finally climbed back to 0.3 seconds when only the good circuit was being simulated. Considering that simulating a single circuit requires about 0.3 seconds per pattern, the effective benefit of simulating all of the circuits concurrently starts at 6 times serial simulation, rises to 60 times, and drops back down to 1.

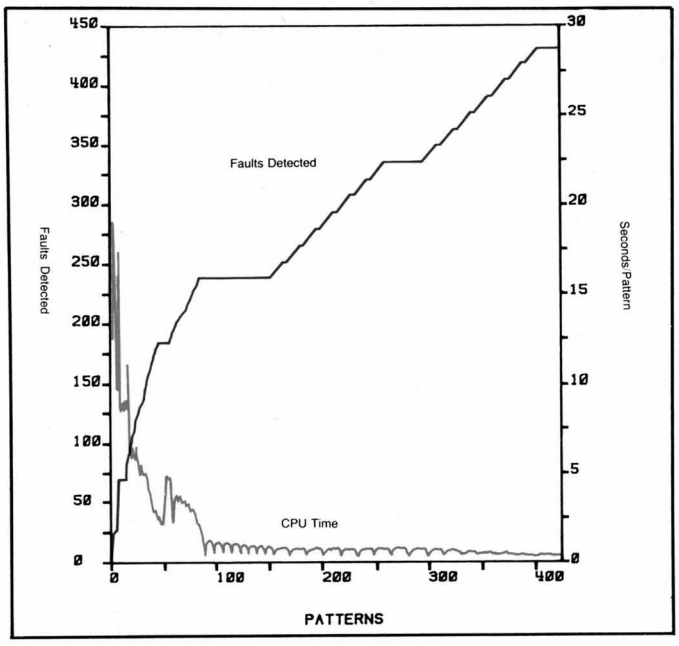


FIGURE 3. Performance of FMOSSIM on a memory circuit.

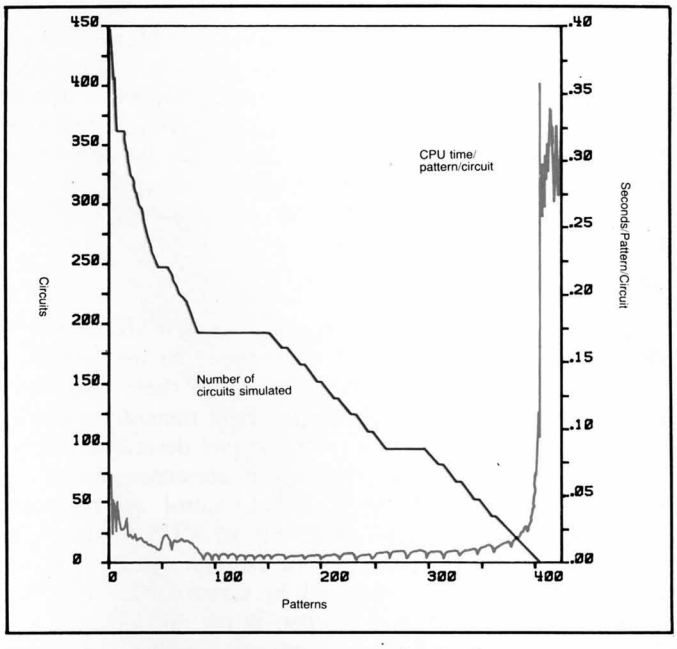


FIGURE 4. Effective concurrency for test case.

Over the entire test sequence, simulating the good machine alone required 2.5 CPU minutes. Our fault simulation required 11 CPU minutes, whereas simulating each faulty circuit serially until it produced a different result than the good circuit produced would take almost 6 hours. Thus, in this case, concurrent simulation has a thirty-fold net advantage over serial simulation. Such a performance gain is clearly well worth the effort.

Conclusion

Our initial experience with FMOSSIM has shown it to be very useful in developing test sequences, especially for novice test designers. Even when developing a test for a small section of an integrated circuit (such as an ALU or a register array), the fault simulator provides information that is hard to obtain by any other means. It quickly directs designers to the areas of a circuit that require further tests. We expect that this type of simulator will become a standard tool for the MOS designer. □

References

Bose, A., P. Kozak, C-Y Lo, H.N. Nham, E. Pacas-Skewes, and K. Wu. July 1982. "A Fault Simulator for MOS LSI Circuits," *19th Design Automation Conference Proceedings*, Las Vegas, NV.

Bryant, R. Fourth Quarter 1980. "An Algorithm for MOS Logic Simulation," *LAMBDA*.

Bryant, R. July 1981. "MOSSIM: A Switch-Level Simulator for MOS LSI," *18th Design Automation Conference Proceedings*, Nashville, TN.

Bryant, R., M. Schuster, and D. Whiting. March 1982. *MOSSIM II: A Switch-Level Simulator for MOS LSI, User's Manual*, Technical Report 5033, Department of Computer Science, California Institute of Technology.

Bryant, R. January 1983. *A Switch-Level Model and Simulator for MOS Digital Systems*, Technical Report 5065, Department of Computer Science, California Institute of Technology.

Galiay, J., *et al.* June 1980. "Physical versus Logical Fault Models MOS LSI Circuits: Impact on Their Testability," *IEEE Transactions on Computers*.

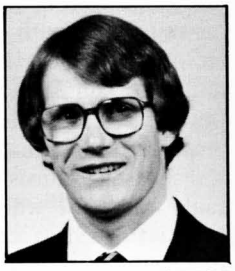
Ulrich, E. and T. Baker. June 1973. "The Concurrent Simulation of Nearly Identical Digital Networks," *Design Automation Workshop Proceedings*, (also *IEEE Computer*, April 1974.)

Wadsack, R. May/June 1978. "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits," *Bell System Technical Journal*.

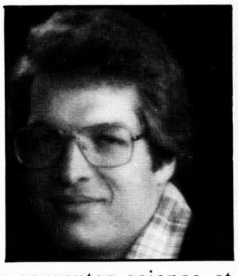
Winegarden, S. and D. Pannell. 1981. "Paragons for Memory Test," *International Test Conference*, Philadelphia, PA.

Xidak, Inc. 1982. *MANSAIL Language Manual*. Menlo Park, CA.

About the Authors



Randy Bryant received the B.S. degree in applied mathematics from the University of Michigan in 1973, and the S.M. (1977), E.E. (1978), and Ph.D. (1981) degrees in electrical engineering and computer science from M.I.T. Since 1981, he has been an assistant professor of computer science at the California Institute of Technology, teaching courses in computer architecture and switching theory, and conducting research in switch-level models of MOS circuits.



Michael D. Schuster received the B.A. degree in applied physics and information science from the University of California at San Diego in 1977. He then joined Burroughs Corp., where he was a project leader in the semiconductor design aids group. He represented Burroughs on the Silicon Structures Project at the California Institute of Technology during the 1981-1982 school year. He is currently pursuing the Ph.D. degree in computer science at Caltech in the areas of modeling, simulation, analysis, and testing of MOS integrated circuits.