Special Applications of the Voting Model for Bridging Faults

Steven D. Millman, *Member, IEEE,* and John M. Acken, *Member, IEEE*

Abstract—**A transistor-level examination of bridging faults and the resulting logic-level bridging fault model is described. Experiments with simulations and silicon demonstrate its accuracy. Previous work has demonstrated the accuracy and efficiency of the voting model for bridging faults. This paper presents a complete formal description of the voting model. In addition, simple solutions to special applications of the voting model which generalize its applicability are presented. These applications include the Byzantine General's Problem, complex gate designs, and nonuniform transistor sizes. How to use Binary Decision Diagrams to compare the voting model to other fault models is also presented. Finally, delay tests are shown to be a poor means for detecting bridging faults. This work was done to show that the voting model, a logic-level model for bridging faults, accurately describes the behavior of real faults in real circuits.**

I. Introduction

THE accuracy of testing analysis depends upon an accurate behavioral description of the circuits on chips containing physical failures. Because tests are applied to chips containing physical failures, the fault models used to grade and generate test sets must be evaluated with respect to defective chip behavior. Therefore, one must concentrate on fault models that help identify chips containing the most likely types of failures. Previous work shows that the greatest number of defects cause bridging faults [1]–[3]. It has also been shown that using inadequate bridging or stuck-at fault models can lead to poor results in both detection and diagnosis of bridging faults [4]. Hence, it is essential to accurately model bridging faults.

II. The Voting Model

Bridging fault models describe the circuit behavior due to the physical failures that cause signal nodes to be shorted together. Each signal node is the output of a different gate. Each of the gates attempts to drive the shorted nodes to a logic value equal to the gate output in the fault-free case. In CMOS, each node is represented as a capacitive load being charged to a voltage level (and, hence, a logic value) by the current the gate output supplies. For shorted nodes, two circuits drive a load equal to the sum of the two independent load capacitors. When the circuits drive in opposite directions, the difference in the output currents charges the load. Therefore, the driver providing the greatest current determines the logic value of the shorted nodes.

The drive of each transistor is a function of the fabrication process, the type of doping, and the size of the transistors. The I–V characteristics are such that transistors behave as voltage-controlled current sources. That is, if the input voltage is large enough, the transistor provides current to the output node. The current supplied by a transistor is proportional to its W/L ratio, and may be represented as a nonlinear resistance between the source and drain when the device is turned on. When the voting model was originally introduced, the term "equivalent resistances" was used. Although a clear statement was made that this was not for timing or voltage calculations, the word "resistance" lead to model misuse. The term "strength" could be used, but many logic models use that term to represent discrete steps for strengths of signals. Because the essence of the voting model is the relative strength of the shorted circuits, the term "relative puissance," RP, will be used (puissance is a synonym for strength.)

The path leading to either the Vdd or the Ground nodes with the greatest relative puissance represents the greatest current drive and wins the vote to determine the logic value on the shorted nodes. The relative puissance is only used to compare the pull-up path to the pull-down path; it is not used to calculate voltages or time constants. The relative puissance is calculated from two sources of information. One source is static and is the number, type, and size of transistors connected to a given node. This source is determined by the technology and layout of the circuit, which are constant during testing. The other source varies with time and tells which transistors are turned on as a result of the application of each vector.

The voting model has different regimes of operation for different circuit conditions. When one of the gates driving the shorted nodes is stronger than the other gate under all conditions, its output dominates the other gate and determines the resulting logic value on the shorted nodes. There is a dominant node regime for each of the two nodes involved in a short, in which one of the nodes always wins the vote regardless of the logical values on the nodes. Another regime is the Wired-AND regime, in which the pull-down networks driving the shorted nodes always overpower the pull-up networks. The fourth case is the Wired-OR regime, in which the pull-up networks always win the vote. It is therefore seen that the classical wired-AND and wired-OR models for bridging faults are subsets of the voting model. Finally, the most complex regime is the one in which the winning network is a function of the inputs of the networks driving the shorted

Manuscript received August 2, 1993; revised September 30, 1993.

S. D. Millman is with Motorola, Inc., 2100 E. Elliot Rd., MD EL510, Tempe, AZ 85284.

J. M. Acken is with Intel, Corp., 2200 Mission College Blvd., MS RN4-39, Santa Clara, CA 95052.

IEEE Log Number 9214778.

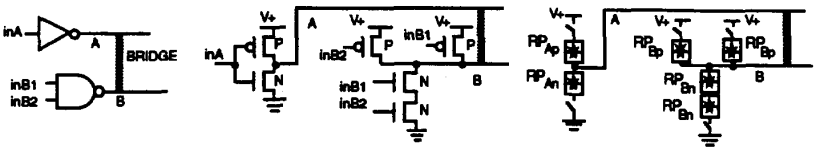


Fig. 1. Bridging fault between an inverter and a NAND gate to demonstrate the voting model.

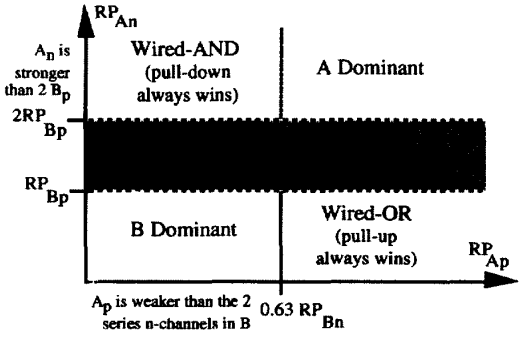


Fig. 2. Voting model regimes for NAND gate–inverter bridging fault.

nodes. The inputs determine not only which logic value results from the circuits, but the number of paths which drive the output to that value.

As an example, consider the output of an inverter bridged to the output of a NAND gate as in Fig. 1. Fig. 2 shows the conditions under which the wired logic and dominant node regimes of the voting model occur, and the central shaded area denotes the most complex regime. In this regime of operation, the logic value of the shorted nodes is not only a function of the gate outputs but also a function of the gate inputs. In particular, the number of inputs to the NAND gate having a logic value of 0 determines the relative puissance of the pull-up network. This regime occurs when the relative puissance of the inverter pull-down network is between the relative puissances that the NAND gate pull-up network can achieve, i.e., $RP_{Bp} < RP_{An} < 2RP_{Bp}$. This is a common situation in CMOS. Table I shows the node values for the fault-free and faulty circuits in this regime of operation. The table shows two situations in the shorted output columns: one where the inverter pull-up transistor is able to outvote the NAND gate pull-down network, and the other where it is not. (In the technology used for this study, two n-channel transistors have 63% the relative puissance of a single n-channel transistor. Hence, the 0.63 in Table I and Fig. 2.) For both shorted output columns $RP_{Bp} < RP_{An} < 2RP_{Bp}$.

In summary, the voting model compares the relative puissances of the devices attempting to pull the shorted node high (toward a 1) to the relative puissances of the devices attempting to pull the shorted node low (toward a 0). The devices with the greatest combined puissance "win" the vote, and the logic value toward which those devices are attempting to drive is assigned to the shorted node. The resulting values can be used to create a fault dictionary for diagnosis.

TABLE I
OUTPUT VALUES FOR NAND-INVERTER SHORT IN COMPLEX REGIME

	Inputs			Outputs		Shorted Outputs	
Line	in$A$	in$B1$	in$B2$	A	B	$RP_{Ap} > 0.63RP_{Bn}$	$RP_{Ap} < 0.63RP_{Bn}$
0	0	0	0	1	1	1	1
1	0	0	1	1	1	1	1
2	0	1	0	1	1	1	1
3	0	1	1	1	0	1	0
4	1	0	0	0	1	1	1
5	1	0	1	0	1	0	0
6	1	1	0	0	1	0	0
7	1	1	1	0	0	0	0

III. CMOS—How It Works

When shorted nodes are attempting to drive to opposite values, the resulting voltage is unlikely to be 0 or 5 V, but will be between these values. In the past, these voltages have been considered indeterminate, unknown values. All electronic circuit behavior depends upon analog node voltages. Within digital circuits, ranges of these voltages are interpreted as logic levels. Frequently, voltages are defined for the logic levels that leave a gap between the level defined to be logic 0 and the level defined to be logic 1. This section describes how circuits resolve the voltages into one or the other logic value.

CMOS logic gates have high voltage gain in the narrow transition region and low gain in the wide stable logic level regions (see Fig. 3). This example is for a p/n ratio such that the current drive of the p-channel transistor is approximately equal to the current drive of the n-channel transistor. The unity gain points are approximately 2.1 and 3.1 V. The logic threshold, or switching point, is the point at which the input voltage equals the output voltage. For this example, the logic threshold is 2.5 V. The final judge of the logic level on any given input to a logic gate is the logic gate itself. In the absence of noise, this implies that any input voltage above the logic threshold is a logic 1, and any voltage below is a logic 0, which leaves no range for intermediate logic levels.

Understanding how CMOS determines logic values from voltages is only half the battle; the voltages created by bridging faults are also important. Fig. 4 shows the output voltage of a short as a function of the ratio of p-channel size over n-channel size for a particular technology. This plot indicates that ambiguous voltages only occur in the range of ratios from 1.6 to 1.9. Although this ambiguous value can be determined from the circuit design, this narrow size indicates that a short in that range is unlikely.

The behavior of a circuit containing shorts is dependent upon the resistance of the short relative to the resistance of other parts of the circuit. Yarbrough [5] designed and

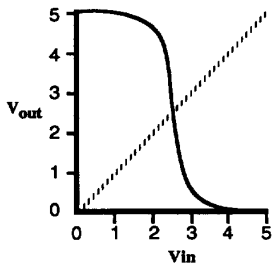


Fig. 3. An inverter transfer curve.

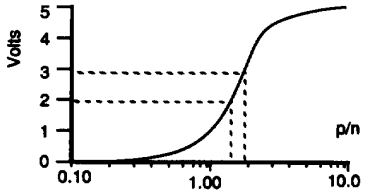


Fig. 4. Bridging fault voltage as a function of p-channel size to n-channel size.

tested structures for process yield monitoring and failure diagnosis. Data from his measurements provide the resistance distributions used in analysis of the voting model. Resistances greater than 4 kΩ can be ignored because they do not degrade the voltage level on either of the shorted nodes beyond logic threshold levels. Those shorts with resistances between 2 and 4 kΩ require analysis to predict the logic values that will result due to a short in a real circuit. Resistances less than 2 kΩ allow the shorted nodes to have equal logic values. The resistance of shorts is broken into three ranges: greater than 4 kΩ, in which the logic values of the shorted nodes remain unaffected; between 4 and 2 kΩ, in which very few actual shorts occur; and finally lower than 2 kΩ, in which the voltage results in equal logic levels on the node. More recent measurements demonstrate a similar distribution of low, medium, and high impedance shorts [6]. This defines the applicability limit for the voting model: the voting model is accurate for all shorts except the rare case when the resistance of the short is between 2 kΩ and 4 kΩ.

An example involving more than one pair of nodes demonstrates the voting nature of a bridging fault. The bus out of a test RAM was driven by separate, identical inverters (see Fig. 5). The drive of each n-channel device was slightly more than twice that of each p-channel device. The circuit produced a 1 on all twelve of the shorted bits ($b1$–$b12$) whenever any nine or more had been written as 1; however, when four or more of the bits were written as 0, all twelve were 0. This example also shows the unlikelihood of indeterminate logic values in CMOS circuits. Because twelve lines were shorted, each input combination represents a 1/12 variation in relative puissances between n-channel and p-channel transistors. Even with the 1/12 steps, solid logic values were read in all cases.

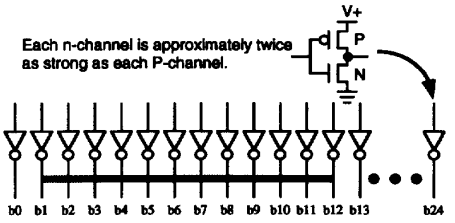


Fig. 5. RAM data bus with bits 1–12 shorted.

In summary, CMOS digital circuits usually resolve all node voltages into full scale logic values as predicted by the voting model. Neither variations in transistor threshold voltage nor variations in the resistance of shorts significantly degrade the accuracy of the voting model. The variations in logic threshold and transistor relative puissances are very important for the voting model, and these can be extracted from layout and process information. Fault models are most frequently used in fault simulation, but the models also form the basis for many testing concepts. A more accurate fault model provides the correct assumptions for test generation and measures of testability.

IV. Generalizing the Voting Model

When the outputs of two gates are shorted together and the gates are attempting to drive to opposite values, the resulting voltage is rarely 0 or 5 V; rather, the resulting voltage lies between these values. The above section and previous work have demonstrated that such values nearly always result in voltages that are sufficiently distant from threshold voltages and that the gates whose inputs are driven by the outputs of the shorted gates interpret the voltages cleanly [7]. However, a problem may arise if one of these downstream gates interprets that voltage as a logic 0 while another interprets it as a logic 1. This problem, the semiconductor industry's version of the Byzantine General's Problem, was solved in [8]. That solution is now generalized to allow for parameter drift due to process variation.

For the following examples, a simple library consisting of an inverter, a 2-input NOR, and a 2-input NAND will be analyzed. For all of the logic gates, all p-channel transistors are the same size and all n-channel transistors are the same size. For this analysis, an expected process variation of 10% was used. (For a different process variation, a different percentage should be used.) The library was created to illustrate the process which follows and was not intended to reflect the entries in any commercially available library. The process for generating the required information is shown in Fig. 6. Each of the steps in this process will now be described.

1) The first step uses a circuit-level simulator, such as SPICE, to calculate all of the puissances for each gate. This is done by simulating all of the possible n-channel paths and normalizing the amount of current each path sinks to the amount of current a single, minimum-sized n-channel transistor sinks. The same process is done for the p-channel transistor paths, normalizing the current each path sources to the amount of current a single, minimum-sized n-channel

TABLE II
PUISSANCE ANALYSIS

Path	Puissance	Path	Puissance
nn	0.67	pp	0.63–0.57
n	1.00	p	1.17–1.05
$n2$	2.00	$p2$	2.34–2.11

TABLE III
THRESHOLDS

Gate-Input	min	max
inverter	2.45	2.50
NAND2-$i1$	2.47	2.53
NAND2-$i2$	2.63	2.69
NOR2-$i1$	2.38	2.44
NOR2-$i2$	2.19	2.25

TABLE IV
BRIDGING FAULT VOLTAGES

	nn	n	$n2$
pp	1.71–1.96	0.82–0.95	0.35–0.40
p	3.29–3.52	2.02–2.40	0.72–0.83
$p2$	4.26–4.34	3.83–3.97	2.02–2.40

transistor sinks. The numbers computed in this manner are the relative puissances of the transistor paths. In order to handle process variation, a range of puissances is created for the p-channel transistor paths. This is done by modifying each p-channel transistor relative puissance by $\pm\Delta RP_p$, where Δ is chosen to reflect the expected process variation's affect on the ratio of p-channel to n-channel conductance, and RP_p is the relative puissance of the p-channel transistor path. The n- and p-channel transistor paths are determined by the gate designs in the library. In addition to the paths, the process parameters are also required for this step. For the example library, the possible p-channel paths are a) a single transistor, p; b) two in series, pp; and c) two in parallel, $p2$. The possible n-channel paths are a) a single transistor, n; b) two in series, nn; and c) two in parallel, $n2$. Table II shows the Puissance Analysis Table which results from this step.

2) The second step consists of using a circuit-level simulator to generate a table of threshold voltages for each input of each gate. The various gate designs and the process parameters are required for this step. The analysis is done for the extremes of the process variation. Table III shows the Threshold Voltage Table for the example library of gates.

3) The third step uses the circuit-level simulator to generate the Bridging Fault Voltage Table. To fill in the table, a simulation is done for each possible p-channel transistor configuration trying to pull a node toward the power rail, in this case 5 V, for each possible n-channel transistor configuration trying to pull the node toward ground. The resulting node voltage for each simulation is placed in the table. Table IV shows the Bridging Fault Voltage Table for the example library. As above, the analysis is done for the extremes of the process.

4) The fourth step is to build the Bridging Fault Results Table. Each column corresponds to one of the possible p-channel transistor paths shorted to one of the possible n-channel transistor paths. Each row corresponds to one of the inputs of one of the gates in the library. To obtain each entry, the threshold voltage of the input for that row, which is taken from the Threshold Voltage Table, is compared to the bridging fault voltage for that column, which is taken from the Bridging Fault Voltage Table. If, for the full range of variation of the process, the threshold voltage is always less than the bridging fault voltage, a P is entered. This indicates that this input interprets the bridging fault such that the p-channel transistor configuration outvotes the n-channel transistor configuration. Likewise, if the threshold voltage is always greater than the bridging fault voltage, an N is entered. Otherwise, a "—" is entered. This does not mean that the value of the bridging fault is indeterminate; rather on some die the bridging fault will be interpreted as a logical 0, and on other die the same bridging fault will be interpreted as a logical 1. Table V shows the Bridging Fault Results Table for the example library.

TABLE V
BRIDGING FAULT RESULTS TABLE

	pp-nn	pp-n	pp-$n2$	p-nn	p-n	p-$n2$	$p2$-nn	$p2$-n	$p2$-$n2$
Inverter	N	N	N	P	N	N	P	P	N
NAND2-$i1$	N	N	N	P	N	N	P	P	N
NAND2-$i2$	N	N	N	P	N	N	P	P	N
NOR2-il	N	N	N	P	N	N	P	P	N
NOR2-$i2$	N	N	N	P	—	N	P	P	—

TABLE VI
PUISSANCE RANGES

	DP	DN
Inverter	0.50	0.11
NAND2-$i1$	0.50	0.11
NAND2-$i2$	0.50	0.11
NOR2-$i1$	0.50	0.11
NOR2-$i2$	0.50	-0.10

5) The last analysis in the process builds two tables. The first, the Puissance Range Table, has one row for each input of each gate, and two columns. The first column contains the smallest difference between puissances that results in that input interpreting the bridging fault voltage as a logical 1. This difference is formed by subtracting the puissance of an n-channel transistor configuration from that of a p-channel transistor configuration. For this computation, the largest value of the range of the puissance of the p-channel transistor configuration is used. The differences are calculated for all of the possible bridging fault paths. The smallest difference that always corresponds to a P in the Bridging Fault Results Table is entered in the first column (DP). The second column of the Puissance Range Table contains the largest difference between puissances that results in that input interpreting the bridging fault as a logical 0. For this computation, the smallest value of the range of the puissance of the p-channel transistor configuration is used (DN). Table VI shows the Puissance Range Table for the example library. Note that the puissance ranges are the same for both of the NAND gate inputs because of where their threshold voltages were located relative to the bridging fault voltages, not because of any type of symmetry.

6) The second table built in the last process step is the Byzantine Table. This table contains an entry for each of the cases in step 5 where the difference of the relative puissances is

TABLE VII
BYZANTINE TABLE

	pp-nn	p-nn	p-n	$p2$-$n2$
Inverter		P	N	N
NAND2-$i1$		P	N	N
NAND2-$i2$		P	N	N
NOR2-$i1$		P	N	N
NOR2-$i2$	N	P	—	—

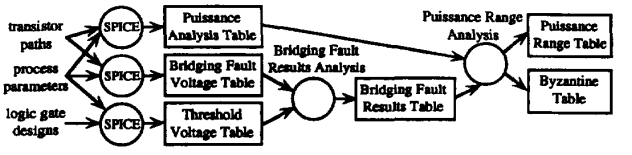


Fig. 6. Process flow for generating voting model tables.

between the smallest difference that always results in a logical 1 and the largest difference that always results in a logical 0. The entries in this table are the same as those in the bridging fault results table except that only those entries meeting the criterion are included. Table VII shows the Byzantine Table for the example library. (Blank entries, such as those in the pp-nn column, do not meet the criterion and therefore need not be stored in the table.)

The process shown in Fig. 6 results in three tables that are used in fault simulation and test pattern generation. The three tables are the Puissance Analysis Table, the Puissance Range Table, and the Byzantine Table. Note that these tables grow linearly with the number of gates in the library and will therefore be of manageable size.

For the example library, there are two "—" values in the Byzantine Table. This illustrates that different inputs of the same gate can interpret the result of a bridging fault differently: input 1 of the NOR2 interprets all of the possible bridging faults as either N or P, whereas input 2 has a "—" for the p-n and $p2$-$n2$ bridging faults. Note that a gate input can interpret the n-channel transistor path as the winner of the vote even if the relative puissance of the p-channel transistors is larger than that of the n-channel transistors. An example of this situation is the input to the inverter in the case of a p-n bridging fault. It is also possible for a gate input to interpret the p-channel transistor path as the winner even if the relative puissance of the n-channel transistors is larger than that of the p-channel transistors.

During fault simulation, the value to be propagated is computed as follows. Given the inputs to the gates whose outputs are shorted, the puissances of the n-channel and p-channel transistor paths are obtained from the Puissance Analysis Table. (Obviously, this computation is only done if the gates drive to opposite values in the fault-free circuit.) The difference between the relative puissances is computed. Then, for each input of each gate to which the shorted nodes fan out, the difference is compared to the input's entry in the Puissance Range Table. If the difference is larger than the entry in the first column, then the input interprets the result of the bridging fault to be a logical 1. If the difference is less than the entry in the second column, then the input interprets the result of the bridging fault to be a logical 0. If the difference is between the entries in the two columns, then the way in which the input interprets the bridging fault is found in the Byzantine Table. If the entry in the Byzantine Table is "—", then the fault cannot be detected through a path containing this gate input.

During test pattern generation, propagation of the fault is handled in the same way as in fault simulation. Deciding on goals—such as how many and which inputs of the gates whose outputs are shorted should be set to 0 or 1—is done by examining the same three tables. The decisions are made in order to force the gates to drive their outputs to opposite states such that the effect of the fault is propagated via a fanout that is interpreted differently than in the fault-free circuit. The voting model can be used for both nonfeedback and feedback faults. In the case of feedback faults, the voting model accurately predicts when state retention or oscillation will occur [9].

Although a small percentage of the possible bridging faults are expected to result in "—" values in the Byzantine Table, several means of eliminating them are available. The first is to add a few simple design rules. These rules would be as follows. 1) Do not use gates that have "—" values in the Byzantine Table. 2) If Rule 1 must be violated, do not connect the gate inputs with "—" values to the outputs of gates that can cause those "—" values. For example, if $p2$-$n2$ causes a "—" for a gate input, that gate input should not be driven by any gates which can have two parallel p-channel transistors or two parallel n-channel transistors driving their outputs.

The above rules can be violated if it is known that the bridging fault that may cause problems can not occur. For example, consider the case where the output of a NAND2 is to be connected to a gate input where the $p2$-$n2$ bridging fault results in a "—" value. If it is known that, for this particular NAND2 gate, that the output of the NAND2 cannot short to the output of a gate that has a $p2$ transistor path, then the "—" value in the Byzantine Table will not be encountered. While these rules may, at times, be easy to enforce, they may not always be practical. In that case, a second approach can be used.

The second approach to ridding the table of "—" values foregoes adding design rules, and instead changes the gate designs so that the problem bridging faults no longer result in a "—". While this is a simple solution, it has the drawback of changing the thresholds, and therefore the performance characteristics, of some of the gates in the library. In addition, entries that had been N or P in the Bridging Fault Results Table may now become "—".

A third, more pragmatic, approach can also be taken. This approach asks the designer to attempt to avoid the situations leading to "—" values in the Byzantine Table. For those cases where the situations cannot be avoided, either use another path to propagate the effect of the fault to an output or use another combination of inputs to the gates driving the shorted nodes to detect the fault. For example, if the short is between the outputs of a NAND2 and a NOR2, try to propagate the affect of the fault to an output through a fanout other than input 2 of a NOR2 since another gate may not have a "—" value for this fault. If this other fanout is sensitized to an output, the

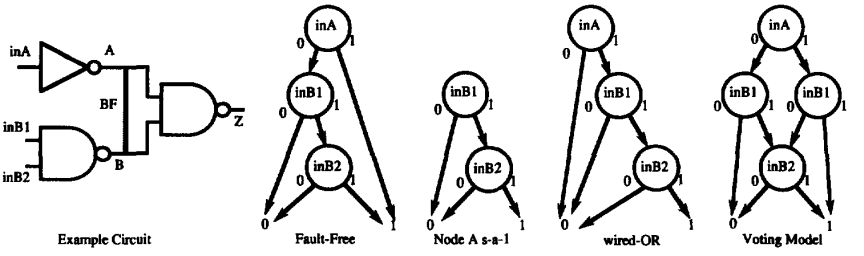


Fig. 7. BDD's for the fault-free circuit, a stuck-at fault, a wired-OR fault, and the voting model.

fault will be detected despite the "—" value propagated by one or more fanouts. If this is not possible, use inputs other than two 0's and two 1's or one 0 and one 1 to the inputs of the shorted NAND2 and NOR2, respectively. That is, attempt to activate and propagate the fault using one 0 and two 1's or two 0's and one 1. In this manner, the resulting logic value will be propagated as the same value on all chips. If only "—" values are propagated, then the bridging fault is undetectable using logic-level tests.

The techniques outlined above show that the voting model can handle the Byzantine General's Problem in the presence of parameter variation. Note that all of the circuit-level simulations are done only once and that they are done before any fault simulations or test pattern generations are attempted. Hence, only logic-level simulations are required during test pattern generation and fault simulation. The following sections detail how the voting model handles complex gates and transistor size variation.

V. Modeling Complex Gates

Since circuits seldom contain only inverters, NAND's, and NOR's, the ability of the voting model to handle complex gates is critical to its success. Simply put, the complex gates are analyzed in exactly the same manner as simple gates. The thresholds of the gates are obtained and added to the Threshold Voltage Table. Then, any new p-channel and n-channel transistor paths are analyzed for addition to the Puissance Analysis Table. Then the voltages which result from shorts to the other n-channel and p-channel transistor paths in the library are computed and added to the Bridging Fault Voltage Table. Finally, the Bridging Fault Results Table, the Puissance Range Table, and the Byzantine Table are regenerated.

VI. Nonuniform Transistor Sizes and Full-Custom Designs

The above examples used a library with uniform transistor sizes. However, this is not a requirement. In fact, the voting model can handle a library with nonuniform transistor sizes just as easily. If nonuniform transistor sizes are used, each unique p-channel transistor path is handled individually, that is, a single transistor of width $W1$ would be analyzed separately from a single transistor of width $W2$. For example, each would have its own row in the Puissance Analysis Table and the Bridging Fault Voltage Table. Hence, no modifications are required to the algorithm already described. Since the three tables required for test pattern generation and fault simulation, namely, the Puissance Analysis Table, the Puissance Range Table, and the Byzantine Table, all grow linearly with the size of the library, even a large library is easily manageable.

In the case of a full-custom design, each of the unique gates in the circuit is treated as a gate in the library, and the library is then analyzed. Hence, the voting model can be used in a practical manner to accurately model bridging faults even for today's largest designs.

VII. BDD Representations of Bridging Faults

Previous work has been done comparing the voting model to other models using analysis [2], [4], [8], [10], current measurements [11], and simulation [7], [12]. Binary Decision Diagrams (BDD's) can be used to analyze Boolean problems [13]. Work comparing fault models using BDD's have used stuck-at, wired-AND, and wired-OR fault models [14]. The voting model is more accurate than these fault models, and because it is a logic-level model, it is easily represented with BDD's. Fig. 7 shows a circuit with several related BDD's. The voting model BDD is for the case where the relative puissance for a single p-channel transistor (RPp) is less than the relative puissance of a single n-channel transistor (RPn) or two n-channel transistors in series ($RPnn$). The relative puissance of an n-channel transistor is less than that of two p-channel transistors in parallel ($RPp2$). A quick glance reveals that stuck-at faults simplify the BDD. This is intuitively correct since a stuck-at node reduces the number of value choices (or binary decisions) the circuit can make; the BDD is therefore simplified. A BDD for the wiredAND case is not shown, because a wired-AND between inputs to a NAND gate does not affect circuit operation. In the past, some have suggested that shorted nodes can be adequately modeled by using both wired-AND and wiredOR fault models. Comparing the voting model BDD to a combination of the wired-OR BDD and the wired-AND BDD (which is the same as the fault-free BDD in this example), one can see that the combination cannot represent the voting model and therefore is unable to accurately model the bridging fault in this example.

Two conclusions can be drawn from the BDD representations of the voting model. The first, and most important, is that the voting model is a logic-level model. This is demonstrated by the straightforward BDD, not elaborate simulation or exotic

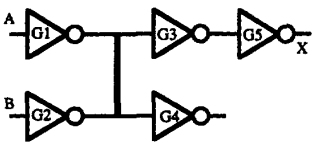


Fig. 8. Test circuit for modeling bridging faults as delays.

TABLE VIII
THE EFFECTS OF A BRIDGING FAULT ON DELAY

Circuit Configuration	Delay
fault-free delay; nodes driving either direction	0.8 ns
faulty delay; nodes driving same direction	0.5 ns
faulty delay; nodes driving opposite directions	2.6 ns

algebras. Therefore, all of the analysis and algorithms that rely upon BDD's can be applied to the voting model. The second conclusion that can be drawn is that the voting model cannot be simply represented by some combination of other models. Other research has demonstrated the accuracy of the voting model, and this shows the availability of standard BDD analysis for the voting model. These two conclusions combine to require inclusion of the voting model in any fault model comparisons.

VIII. THE DELAY EFFECTS OF BRIDGING FAULTS

A bridging fault can have an electrical effect on a circuit even if the circuit does generate the correct response to the input stimulus. For example, when a gate wins the vote, the final outcome of the vote is delayed since it must overcome the other gate. Therefore, bridging faults can result in the same kinds of effects as delay faults. Prior work has been done on the delays bridging faults can introduce and how to detect them using these delays [15].

Table VIII shows the effects of a bridging fault on delays. The data are for a short between the outputs of two inverters. The circuit used for the experiment is shown in Fig. 8. The delays shown in Table VIII were measured from input A to output X. During the simulation, input B was held at 0. As a result, when A changed from 0 to 1, inverter $G1$ had to overcome inverter $G2$ to pull down the shorted nodes. As can be seen, this more than tripled the delay through the circuit. On the other hand, when input A changed from 1 to 0, inverter $G2$ helped pull the shorted nodes high resulting in over a 30% decrease in delay.

Clearly, a good delay test could be used to detect this bridging fault if the first pattern sets the shorted nodes to the same value and the second pattern changes one of them to the opposite value. However, detecting bridging faults via delay fault testing is not practical for several reasons. First, the tests would have to be applied at speed, which requires either extensive built-in self-test or expensive testers. Second, a delay test consists of two patterns where the second pattern must be applied immediately after the first. This implies that shift registers with shadow registers must be used in a scan-based design. Such registers require more space than shift registers without shadow registers, and therefore require more overhead. In large designs, the additional area penalty may be unacceptable. Third, only those bridging faults that introduce more delay than there is slack will be detected. (*Slack* is the amount of time which results from subtracting the actual delay of the path in the fault-free circuit from the period of the clock.) If the bridging fault introduces less delay than the slack for that path, then the bridging fault will not be detected. Hence, only the bridging faults on the longest paths can be detected as delay faults. Thus, the majority of bridging faults would not be detected in this manner.

IX. SUMMARY AND CONCLUSIONS

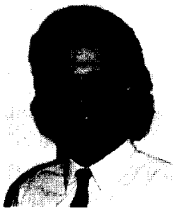
The voting model accurately describes the behavior of shorted nodes in CMOS digital circuits. The logic value of shorted nodes is equal to the logic value output by the circuit with the greatest relative puissance. The probability of an intermediate voltage is very low; therefore, bridging faults result in valid logic values on the shorted nodes. Bridging fault analysis and test pattern generation techniques based upon the voting model are essential for testing chips containing physical failures.

This paper showed that the most accurate bridging fault model, the voting model, could be generalized. In particular, details on how to handle the Byzantine General's Problem, IC fabrication process variations, complex gates, and transistor size variations were presented. An example using BDD's to represent the voting model and to compare it to other fault models was also described. Finally, the relationship between bridging faults and delay faults was discussed, and it was shown that delay tests are poor bridging fault tests. In summary, the voting model is a logic-level model that accurately describes the behavior of real faults in real circuits.

REFERENCES

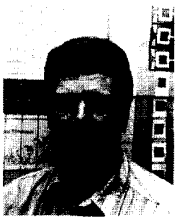
[1] J. Galiay, Y. Crouzet, and M. Vergniault, "Physical versus logical fault models MOS LSI circuits: Impact on their testability," *IEEE Trans. Comput.*, vol. C-29, pp. 527–531, June 1980.
[2] J. P. Shen, W. Maly, and F. J. Ferguson, "Inductive fault analysis of MOS integrated circuits," *IEEE Des. Test*, pp. 13–26, Dec. 1985.
[3] W. Maly, "Realistic fault modeling for VLSI testing," in *Proc. 24th DAC*, Miami Beach, FL, June 28–July 1, 1987, pp. 173–180.
[4] S. D. Millman, J. M. Acken, and E. J. McCluskey, "Diagnosing CMOS bridging faults with stuck-at fault dictionaries," in *Proc. Int. Test Conf.*, Washington, DC, Sept. 10–14, 1990, pp. 860–870.
[5] W. J. Yarbrough, "A testing methodology and test chip design strategy for IC fabrication process assessment, problem diagnosis, and yield analysis," Ph.D. dissertation, Stanford Univ., Stanford, CA, Apr. 1988.
[6] R. Rodríguez-Montañés, E. M. J. G. Bruls, and J. Figueras, "Bridging defects resistance measurements in a CMOS process," in *Proc. Int. Test Conf.*, Baltimore, MD, Sept. 20–24, 1992, pp. 892–899.
[7] J. M. Acken and S. D. Millman, "Accurate modeling and simulation of bridging faults," in *Proc. 1991 CICC*, May 6–9, 1991, San Diego, CA, pp. 17.4.1–17.4.4.
[8] J. M. Acken and S. D. Millman, "Fault model evolution for diagnosis: Accuracy vs precision," in *Proc. 1992 CICC*, May 3–6, 1992, Boston, MA, pp. 13.4.1–13.4.4.
[9] S. D. Millman and J. P. Garvey, "An accurate bridging fault test pattern generator," in *Proc. Int. Test Conf.*, Nashville, TN, Oct. 26–30, 1991, pp. 411–418.
[10] J. M. Acken, "Deriving accurate fault models," CSL-TR-88-365, Comput. Syst. Lab., Stanford Univ., Oct. 1988.
[11] R. C. Aitken, "A comparison of defect models for fault location with Iddq measurements," in *Proc. Int. Test Conf.*, Baltimore, MD, Sept. 20–24, 1992, pp. 778–787.

[12] S. D. Millman, "Nonclassical faults in CMOS digital integrated circuits," Ph.D. dissertation, Stanford Univ., Stanford, CA, Dec. 1989.
[13] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C35, pp. 677–692, Aug. 1986.
[14] K. M. Butler and M. R. Mercer, *Assessing Fault Model and Test Quality.* Dordrect: Kluwer Academic, 1992.
[15] M. Favalli, P. Olivo, and B. Ricco, "Dynamic effects in the detection of bridging faults in CMOS ICs," *JETTA*, vol. 3, pp. 197–205, Autumn 1992.



Steven D. Millman (M'90) was born in White Plains, NY, in 1962. He received the A.B. degree in physics from Occidental College, Los Angeles, CA, in 1984 while concurrently attending the California Institute of Technology. He received the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Palo Alto, CA, in 1985 and 1989, respectively.

He has been with Motorola since 1989 and is the Manager of Large Core Development. His group writes synthesizable behavioral descriptions of previously intractable designs for Motorola's semiconductor sector and equipment divisions. He holds three patents and is a member of the Technical Program Committees of the Custom Integrated Circuits Conference and the Wafer-Scale Integration Conference.



John M. Acken (S'75–M'78) received the B.S. and M.S. degrees in electrical engineering from Oklahoma State University, Stillwater, and the Ph.D. degree from Stanford University, Stanford, CA.

From 1970 through 1973 he served in the Armed Forces as an Electronics Repairman. From 1978 to 1983 he was a member of the Computer-Aided Design Division of Sandia National Laboratories. Since 1989 he has been at Intel, where he is a CAD Development Manager.