Fast and Accurate CMOS Bridging Fault Simulation

Jeff Rearick
Hewlett-Packard
Integrated Circuits Business Division
3404 E. Harmony Rd.
Ft. Collins, CO 80525

Janak H. Patel
University of Illinois
Coordinated Sciences Lab
1308 W. Main St.
Urbana, IL 61801

Abstract

This paper identifies the two key factors involved in obtaining accurate bridging fault simulation results and presents a hybrid technique that maximizes both the speed and accuracy of bridging fault simulation for gate-level standard cell designs. Both combinational and sequential circuits are studied and the results are compared with several other bridging fault simulators.

1 Introduction

The desire to drive the defect levels of integrated circuits to extremely low levels has focused a great deal of attention on the simulation of realistic defects. The traditional single stuck-at fault model does not always correctly predict the behavior of physical defects in contemporary MOS technologies [1, 2, 3]. The impact of the shortcomings in the single stuck-at fault model is that fault simulation using only this model is no longer an accurate estimator of IC quality [4]. The need for more detailed and realistic fault models and associated fault simulators is clearly indicated.

Inductive fault analysis has shown that the most commonly occurring type of fault resulting from fabrication defects, modeled as dust particles of various sizes on photomasks, is the bridging fault [5]; this fault type will be the focus of this work. While much of the early work in bridging faults claimed that either wired-and or wired-or logic resulted when two nodes were bridged [6, 7], recent work has shown that bridging faults cause more complex circuit behavior than cannot be predicted by permanent wired-logic fault models [8, 9]. Simulation methods designed to deal with these complexities exist [10, 11], but can be computationally expensive or limited to combinational circuits. This paper will present a fast bridging fault simulation method that accurately models the behavior of bridging faults by considering circuit-level details near the bridging site while using gate-level simulation everywhere else.

The circuit types targeted in this research are gate-level, synchronous, sequential, digital logic circuits. The bridging defects that are considered in this paper are derived from actual circuit layout data. The vast majority of previous work on bridging faults has been done for combinational circuits with randomly generated bridging fault lists.

The remainder of this paper is organized as follows: the various abstractions of bridging defects into bridging faults will be explored in Section 2, the accuracy of those models is discussed in Section 3, a fast and accurate bridging fault simulation algorithm that has been developed is described in Section 4, and concluding remarks are made in Section 5.

2 Bridging Fault Models

The circuit in Figure 1 will be used to illustrate the differences in bridging fault models at the gate, switch, and circuit level.

Bridging defects are modeled at the gate level as permanent network reconfigurations which alter the logical structure of the circuit. There are four classes of these wired-logic bridging faults: wired-and, wired-or, and two kinds of dominant driver faults (when G always outdrives H (G "wins") and when H always outdrives G (G "loses")). Note that a new node, GH, is created by the addition of the bridging logic gate, and this new node drives all of the former destinations of the bridged nodes G and H. In the case of a dominant driver bridging fault, the losing node is left to dangle, while the winning node is connected to all of the destinations of both nodes. Some IC technologies favor a particular gate-level bridging model; shorts in TTL, for example, typically result in wired-and behavior [12]. Unfortunately, CMOS does not

INTERNATIONAL TEST CONFERENCE 1993

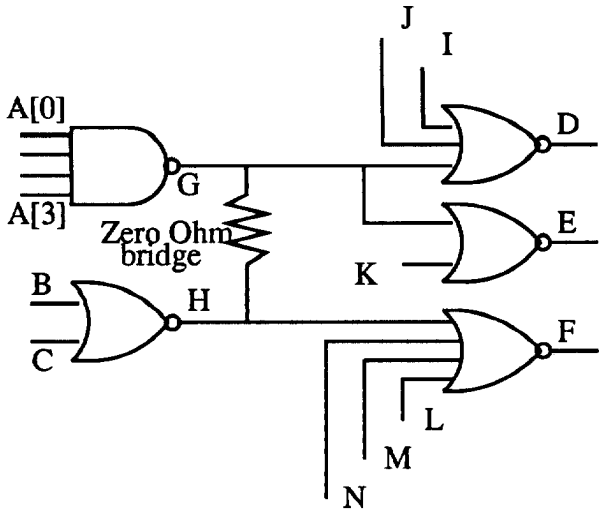

Figure 1: Example circuit with bridge from G to H

Table 1: Gate input pattern impact on output bridge voltage

A[0]	A[1]	A[2]	A[3]	B	C	V_{GH} (V)
0	0	0	0	0	1	4.35
0	0	0	0	1	1	3.68
0	0	0	1	0	1	4.13
0	0	0	1	1	1	3.25
0	0	1	1	0	1	3.68
0	0	1	1	1	1	2.54
0	1	1	1	0	1	2.54
0	1	1	1	1	1	1.38
1	1	1	1	0	0	2.66

adhere to such a fixed behavior, as will be seen.

Switch-level fault simulators, such as COSMOS [13], CHAMP [14], or IDSIM3 [15] can mimic a bridging defect by the insertion of a large, permanently conducting transistor between two nodes or by simply joining the two nodes and forming more complex pullup and pulldown networks. The former method does allow some degree of flexibility in modeling the resistance of the bridge by allowing for different sizes (and hence conductance values) of the bridging transistor. The latter method is simply the degenerate case of the former, where the transistor is infinitely large and has zero resistance.

Circuit-level simulators, such as SPICE2 [16] and SPICE3, can very realistically model a bridging defect by the insertion of a resistor between two circuit nodes. Capacitive and inductive properties of the bridging defect, if known, can also be included, enabling the timing degradation introduced by the defect to be evaluated. While such delay information may be useful in detecting bridging defects, this work focuses only on resistive bridges and their detection by static voltage-based testing.

3 Accuracy of Bridging Fault Models

The chief criterion for evaluating the accuracy of the various bridging fault models will be the degree to which each predicts the correct voltage values on circuit nodes around the bridging fault site for each vector. There are two main factors that differentiate the accuracy of the models: the determination of the voltage value that results at the site of the bridge, and the interpretation of that voltage value by the receivers (the gates downstream of the bridge).

3.1 Bridging voltage determination

The connection of two nodes via a bridging defect implies that the gates driving those two nodes will be engaged in a drive fight (logic contention) when they have opposite values. The permanent wired-logic modifications made to a gate level circuit to model a bridging fault have a major drawback that causes inaccuracy: depending on the inputs to the gates driving the bridge, the logical function of this drive fight can change between wired-and and wired-or behavior. This problem, recognized in [17], is illustrated in Table 1, in which a zero-ohm short exists between the outputs of the NAND and NOR gates of Figure 1 and various input combinations to these gates are applied with the resulting output voltage of the bridged node (V_{GH}) shown. This circuit was constructed using standard library cells and simulated with SPICE3. If this bridged output voltage were being received by a gate input with a logic threshold of 2.9 V, the first five cases would be interpreted as a logic one while the last four cases would be seen as a logic zero. Clearly, the use of a single, permanent wired-logic fault model would produce incorrect results on certain test vectors. The input values to the gates driving the bridge must be considered, not just the logic output values of those gates, to determine the relative drive strengths of the shorted pullup and pulldown networks correctly.

Switch-level simulator evaluation techniques determine node values by comparing the relative strengths

of pullup and pulldown networks and thus take into account the effects of gate input values; the evaluation mechanisms of a switch simulator, designed to handle ratioed logic, are well-suited to correctly predicting the behavior of bridging defects. This "Voting Model" [17] has been shown to be much more accurate than the simplistic gate-level wired-logic bridging fault models, but itself has limitations: non-linear transistors are treated as linear resistors, and the assumption is made that the resistance of the bridging defect is much less than the on-resistance of the transistors. Furthermore, switch level simulation is typically much slower than gate level simulation.

Circuit-level simulation, of course, is capable of extremely accurate simulation, but is even slower than switch-level simulation.

3.2 Bridging voltage interpretation

The second factor that determines the accuracy of a bridging fault model is the interpretation of the voltage at the bridged node by the receiving logic gate(s). The previous section demonstrated that circuit-level details must be considered to deduce the analog node voltages on either side of the bridging defect. Since the desired result for gate-level simulation is a digital value for each node rather than a pair of analog voltages, a conversion must be made back to the digital realm. It is here that the interpretation issue arises.

A CMOS digital logic gate is basically a very high gain amplifier that causes its output to be forced to a power supply rail when the sensitized input is forced above or below its logic threshold. The *input logic threshold* of a gate input is defined as the voltage value at which the input and output of the gate are equal (assuming all other inputs to the gate are held at noncontrolling values). A small deviation of the input voltage above or below the logic threshold, typically a few tenths of a volt, is sufficient to cause a large swing in the output. The decision on how to interpret a bridging voltage must be made with respect to the input logic threshold voltage of each input of each gate downstream from the bridge.

The opportunity for simulator error presents itself at this point because each input of each logic gate can have a different logic threshold, so that two gates tied to the same node can interpret the same input voltage as two different logic values. This situation has been dubbed "The Byzantine General's Problem" in [9], and is illustrated in Figure 2. The SPICE-derived input threshold voltages for the two-, three-, and four-input NOR gates are shown in Table 2. The three-input NOR gate driving signal D in the figure

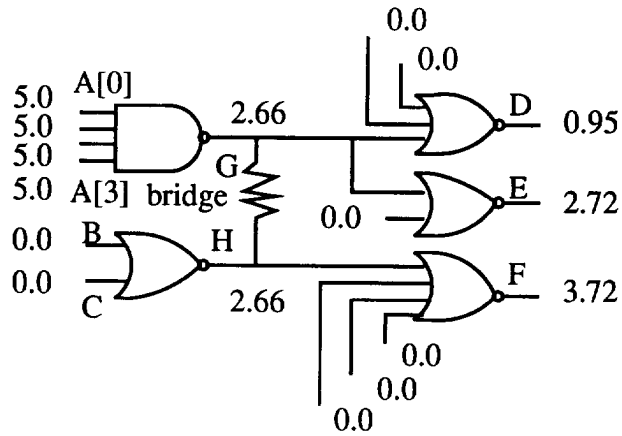

Figure 2: Byzantine bridging voltage interpretation

Table 2: NOR gate input logic thresholds

Gate	Input 1	Input 2	Input 3	Input 4
nor2	2.46	2.66	-	-
nor3	2.46	2.59	2.75	-
nor4	2.55	2.66	2.78	2.90

produces a logic zero because the bridging voltage is about two-tenths of a volt above the threshold of the first (bottom) input, which is sufficient to pull the output down to a voltage (below 1 V) well below the logic threshold of any input of any gate in the library. The two-input NOR gate driving signal E produces an intermediate value since the bridging voltage is so close (about two-thousandths of a volt) to its input logic threshold voltage. It would be safest to classify this value as an "unknown" in the simulation since it lies in the middle of the range of the possible gate input threshold voltages. The four-input NOR gate driving signal F produces a logic one because the bridging voltage is more than two tenths of a volt below the threshold of the last (top) input, thus making it incapable of pulling down the output. The 3.7 V output value is well above the logic threshold of any input for any gate in the library, making this a solid logic one. This example shows that a single node value (the bridged node GH) is interpreted in all three possible (different) ways: 0, X, and 1.

The implication of this behavior is disconcerting: different branches from a single fanout stem can effectively have different logic values! Typical gate- and switch-level simulators use a single value to represent the voltage on each node (stem) in the circuit and thus cannot recognize this behavior without special

modifications. Even then, the restriction to digital values (0, 1, and X) eliminates the possibility of performing comparisons between bridging voltages and input logic thresholds. These restrictions prevent the accurate interpretation of bridging voltages by gate- and switch-level bridging fault simulators.

4 A Fast and Accurate Bridging Fault Simulator

The tradeoff between speed and accuracy in bridging fault simulation techniques should be quite clear by now. Only circuit-level techniques have sufficient information to produce correct results, while only gate-level techniques can perform digital simulation with sufficiently high throughput. This section presents a hybrid approach based on these two techniques that can deliver the accuracy of a circuit-level simulator with the speed of a gate-level simulator.

4.1 Realizations and Ideas

After exploring the mechanics of the various bridging fault simulation methods, these observations can be made:

- the vast majority of time spent in bridging fault simulation is in the digital realm and is very similar to single stuck-at fault simulation,
- circuit-level detail need only be used at the bridging site,
- all likely bridging sites can be identified from layout artwork,
- the only information needed to model the behavior of a bridge correctly is the short-circuit voltage of the two driving gates and the logic threshold of all receiving gates,
- only a limited number of short-circuit voltages are possible based on the identified bridging sites and the number of cells in the library,
- all logic threshold voltages are known from library characterization.

Given these realizations about the nature of the procedures essential to bridging fault simulation, the following ideas suggest themselves:

- use the highest performance single stuck-at simulator available as the underlying core simulator,
- precompute the short-circuit voltage for all combinations of gates (and all pertinent input combinations) that can be involved in a bridging fault,
- precompute the logic threshold voltages for all library cell inputs,
- modify the fault simulator's evaluation routine to specially treat the fault site by using the tabularized short-circuit voltages and logic threshold voltages.

The crux of the new technique is that a single stuck-at fault simulator needs to be extended to perform one additional type of evaluation: determining the digital values at gate inputs that are immediately downstream from the fault site. This is accomplished in two phases:

1. Use the bridging defect's driving gates and their input values to index into a precomputed short-circuit voltage table to obtain the voltage seen by the receiving gates, then
2. Compare that voltage to the precomputed logic threshold voltage for each fanout of the bridge site to obtain a digital value for simulation.

This method does require the up-front use of a circuit-level simulator to characterize the short-circuit and logic threshold voltages of the gate types that will be used in a design. The cost of characterization is a function of the number of gates in the library and the number of inputs per gate. For a library with 100 different gate types and an average of 2.5 inputs per gate, this would translate to 320,000 SPICE runs to determine the bridging voltages and 250 SPICE runs to find the input logic threshold voltages. Fortunately, the simulation models are tiny and run very quickly, the task is easily automated with scripts, and much of the data from the bridging voltage simulations can be discarded since no drive fight is caused by many of the input combinations. The average time per SPICE simulation was found to be 0.363 seconds for this work, which would result in a total characterization investment of only 32 hours for this hypothetical library.

The basic flow of the process of bridging fault simulation is as follows:

1. Characterize the short-circuit voltages of all possible gate combinations in the library.
2. Characterize the logic threshold voltages of all inputs of all gate types in the library.

3. Tabularize these data in the format appropriate for the simulator.

4. For each chip designed in this library:

   (a) Extract the likely locations of bridging defects from the layout data.

   (b) Classify the resulting bridging faults as feedback or nonfeedback.

   (c) Fault simulate the chip given the extracted and classified bridging fault list and a test vector set.

4.2 Differences from previous work

The notion of using precomputed data to determine bridging voltages is not new; Acken and Millman suggested it in [8], Millman and Garvey implemented it in [18], as did Ferguson and Larrabee in [19]. These approaches all have in common, however, the use of the voting model based on the resistance of transistors rather than on explicit SPICE data for each and every case as is done in this work. The importance of gate input logic threshold voltages has also been suggested by these authors, but the use of explicit SPICE-derived data for every gate input is also unique to this simulator. This feature is key to obtaining accurate simulation results. These earlier simulators were also limited to combinational or fully scanned circuits, while this work handles sequential circuits. Greenstein and Patel presented a sequential mixed-level simulator in [10] that executed circuit level simulation when needed during bridging fault simulation that gave good accuracy but at the expense of run time dominated by the circuit simulator. The use of precomputed SPICE data in this work is intended to alleviate that bottleneck.

4.3 Assumptions

The following assumptions were made in the development of the simulator and are responsible for both its speed and its limitations:

- Circuits will be standard cell ASICs.
- Only routing channel bridging faults will be considered.
- The circuits will be represented at the logic gate level.
- The size of the standard cell library is reasonably small (100 gate types or less).

The practical import of these assumptions on the simulator is:

- Only logic gate output shorts need to be handled.
- The storage space for and cost of library characterization is acceptable.

4.4 Simulation Algorithm

In order to maximize fault simulation performance, the differential-based PROOFS algorithm [20] was chosen to be the platform upon which the bridging fault simulator would be built. The basic algorithm had to be extended to perform the following functions:

- inject bridging faults instead of stuck-at faults,
- evaluate the short-circuit voltage at a bridging site,
- compare the short-circuit voltage with the logic threshold voltage of fanout gates to deduce new logical values,
- resolve feedback bridging faults.

Fault injection is accomplished by substituting the gates driving a bridging fault with "fault gates" to indicate that a special evaluation routine must be used at these points. The original gate types are saved as part of the injected-fault data structure. The injection of a bridging fault symbolically ties the outputs of two gates together such that when either one is evaluated, the value at the other is used to modify the output to model the effect of the bridging fault. The output values of the two driving gates are first compared to see if they differ; if not, the bridging fault is not excited and no output change occurs. If the values differ, then a simple table lookup is performed using the gate types of the two drivers as well as their input values to offset into an array of short-circuit voltages.

It is crucial to note that only a relatively small number of entries need to exist in the short-circuit voltage table. The table is organized as a two-dimensional array of "gates which can drive a zero" versus "gates which can drive a one." Only those gate/input combinations which drive with a unique strength need to exist in the table; for the 16 basic gates in the library (NOT, NOR2-3-4, NAND2-3-4, OR2-3-4, AND2-3-4, XOR, XNOR, and DFF) there are 21 unique ways to drive a zero and 21 other unique

ways to drive a one, making a total of merely 441 entries that must be saved in the table. For a hypothetical library of 100 gate types with 2.5 inputs per gate and an exhaustive number of unique ways to drive ones and zeros, there would be approximately 80,000 table entries which, while large, is certainly manageable. An entry in the table is accessed by a function that maps the gate type and number of active inputs for each of the driving gates into the unique row and column numbers that correspond to the short-circuit voltage for that particular drive fight. It should be noted that the actual decimal values of the bridging voltages are not stored in the table; an integer mapping is used to avoid the need for floating-point comparisons.

As Section 2 pointed out, determining the bridging voltage is only half of the work required to accurately model a bridging fault. Once determined, that voltage must be compared with the gate input threshold of the destination. This is accomplished by comparing the integer representation of the bridging voltage to an integer representation of the tabularized logic threshold voltage for each input receiving the bridging voltage. This threshold data is stored in another table that is accessed by using the gate type and the input number to index into an array.

The creation of state storage by feedback bridging faults is addressed by saving the state of the nodes involved in the bridge as if they were architected state storage elements. Since the location of these state elements must be known in advance, the fault list and the circuit database are processed prior to simulation to identify all feedback bridging faults. Then the same mechanisms used for sequential circuit simulation can be applied to these feedback bridging faults. Since some feedback faults can introduce oscillations when properly excited, mechanisms to detect and suppress oscillations were added to the simulator.

4.5 Implementation/Performance

The simulator package consists of a circuit compiler coupled to a bridging fault classifier for sorting faults and identifying feedback bridging faults, as well as the bridging fault simulator (BRIDGESIM). To evaluate the performance of BRIDGESIM, a series of experiments was performed on a subset of the ISCAS combinational [21] and sequential [22] benchmark circuits. Actual layouts of these circuits were created using the OCTTOOLS [23] and the possible bridging defect sites were extracted from the artwork with CARAFE [24]. Single stuck-at test vectors were generated for these circuits using HITEC [25]. Table 3 contains the basic statistics for the eight circuits used in the experiments, including the single stuck-at fault coverage of the tests used as well as a breakdown of the nature of the extracted bridging faults. Since the prototype simulator was constructed to process only nonfeedback faults, the results in this section are restricted to that class of bridging faults.

A bridging fault simulation was run for each of the eight circuits using several different bridging fault simulators:

- EPROOFS [10] is a mixed-level simulator that specifically targets bridging faults. EPROOFS uses circuit-level simulation in the region of the circuit surrounding the bridging fault and gate-level simulation everywhere else. The "analog region" is made large enough to assure that only power-supply rail voltages are passed into the gate-level portions of the circuit.
- FETSIM is a serial, switch-level bridging fault simulator developed during the course of this work which represents bridging faults by physically joining bridged nodes together.
- IDSIM3 [15], is a concurrent bridging fault simulator that uses a matrix algebra-based equation formulation and solution technique. The optimistic algorithm used in FETSIM rarely generates indeterminate values, even in the presence of a bridge, while the very pessimistic algorithm used in IDSIM3 produces many unknown values when bridging faults are introduced.
- PFSIM is a gate-level, unit-delay, event-driven bridging fault simulator that was developed during this research to allow simulation of wired-logic gate-level bridging fault models (AND, OR, WIN, and LOSE). The unit-delay paradigm was chosen because it contains mechanisms that are inherently capable of resolving the feedback loops that can be introduced by bridging faults.

A careful accounting of the accuracy of any given fault simulation technique can be made by checking the classification of each fault against some known reference classification. Since EPROOFS represents the best available technique studied thus far, it was chosen as the reference. Table 4 shows the number of mistakes made by each simulator, with respect to EPROOFS, in classifying the faults in each circuit as detected, undetected, or potentially detected.

The rows "Total" and "Percent" in the table summarize the results to show the percentage of inaccurate classifications made; BRIDGESIM outperforms

Table 3: ISCAS benchmark circuit and bridging fault statistics

Circuit name	# In	# Out	# Gates	# DFFs	SSA Cov%	#Total bridges	#Routing bridges	#Output bridges	#Output FB bridges
c17	5	2	6	0	100	63	34	24	20
c432	36	7	232	0	99.25	2534	1337	1037	730
s27	4	1	10	3	100	229	50	23	17
s208	10	1	104	8	8.29	1408	545	351	192
s298	3	6	119	14	86.04	2107	724	484	195
s344	9	11	160	15	93.86	2300	843	557	288
s349	9	11	161	15	92.57	2338	890	576	308
s641	35	24	379	19	86.51	5003	2605	1877	737
TOTALS	111	63	1171	74	83.32	15982	7028	4929	2487

Table 4: Nonfeedback bridging fault simulation mistake counts, run times (s)

Circuit	EPROOFS	FETSIM	IDSIM3	AND	OR	WIN	LOSE	BRIDGESIM
c17	0	0	4	0	0	0	0	0
c432	0	11	292	30	11	11	11	5
s27	0	2	6	2	3	2	0	0
s208	0	64	69	33	32	33	40	8
s298	0	46	273	23	59	48	31	13
s344	0	44	261	37	41	31	36	21
s349	0	61	260	44	53	30	49	21
s641	0	79	1114	64	84	95	68	55
Total	0	307	2279	233	283	250	235	123
Percent	0	12.57	93.33	9.54	11.59	10.24	9.62	5.04
TOT time	1509.7	2843.2	4643.8	835.1				115.1
AV time	188.7	355.4	580.5	104.4				14.4

all of the alternative simulators but still errs too frequently. The last two rows in the table ("TOT time" and "AV time") are the total and average run times in seconds on an HP 9000-730 for EPROOFS, FETSIM, IDSIM3, PFSIM (which was run four times, once for each of the wired-logic bridging fault models, and totaled), and BRIDGESIM. Notice that BRIDGESIM is faster than the other simulators by roughly an order of magnitude; the goal of creating a fast simulation platform has been met successfully.

4.6 Limitations

While certainly an improvement in both speed and accuracy with respect to other bridging fault simulation algorithms, BRIDGESIM does have weaknesses.

The use of circuits that are completely represented at the gate level and whose bridging faults exist only between gate outputs restricts the applicability of this simulator. There is no guarantee that all likely bridging faults on a any chip will occur only between gate outputs (intragate faults being the chief example). Table 3 clearly shows that only a minority (between a quarter and a third) of actual bridging faults are gate output shorts. It should be noted, however, that shorts between gate outputs tend to be the most probable type of bridge because these signals are usually routed over relatively large areas and are thus prone to spot defects.

The scalability of this simulator can be limited by the size of the lookup table for short-circuit voltages, which can grow geometrically with both the number of gate types and the number of gate fanins. If the number of cells in the library is very large, or if

the number of unique ways to drive ones and zeros is very large, the lookup table for the bridging voltages can become unwieldy. The gate input threshold voltage table grows only linearly with the number of gate types and gate fanins in the library and is not a limiting factor.

The final limitation comes from possible errors in the precomputed data that BRIDGESIM uses. There are basic assumptions made when performing library characterization for the determination of both bridging voltages and gate input threshold voltages that can be violated in certain faulty cases.

The assumption made during the determination of bridging voltages is that all of the inputs of both gates driving the bridge are being driven to full rail voltage values. In most circumstances this is a valid assumption, but when a bridge creates a tight feedback loop that spans only one or two gates, it is likely that the nodes in the loop are being driven to only intermediate values. These intermediate values can then appear on the inputs of a gate that is driving the bridged node, thus violating the assumption that only solid logic values appear at these gate inputs.

The assumption made in the characterization of gate input logic threshold voltages is that only the input being characterized is allowed to change; all other inputs are held at the noncontrolling voltage rail. When a bridging fault occurs between inputs of a single gate, there will be two inputs changing instead of just one, since the bridged nodes are equipotential (in the case of a zero ohm short, and linearly related for nonzero bridging resistances). Such a bridge invalidates the precomputation of the gate input logic threshold.

Another source of error in the precomputed data is an incorrect assumption for the value of the likely bridging resistance. If the resistance of the actual bridging defect is significantly different from that assumed during characterization, the simulator can predict incorrect results. By repeating the SPICE characterization of the short-circuit voltages with a new value of the bridging resistance, a more accurate table can be created for the simulator to use.

5 Concluding Remarks

This work has shown that to predict the behavior of a bridging fault correctly, two key steps to model physical mechanisms must be taken: determining the voltage at the bridging site by resolving the drive fight between the bridged gates, and interpreting that voltage by comparing it to the logic threshold voltages of each gate input connected to the bridged node. The BRIDGESIM program uses precomputed values for these voltages to obtain accurate values to feed into a gate-level simulation platform. This technique delivers excellent throughput and accuracy, but is limited by the scalability of the table lookup procedure and the assumptions made during circuit-level characterization.

References

[1] C. C. Beh, K. H. Arya, C. E. Radke, and K. E. Torku, "Do stuck fault models reflect manufacturing defects?," in *Proceedings of the IEEE International Test Conference*, pp. 35–42, Nov. 1982.

[2] V. V. Nickel, "The inadequacy of the stuck-at fault model," in *Proceedings of the IEEE International Test Conference*, pp. 378–381, Nov. 1980.

[3] A. Pancholy, J. Rajski, and L. J. McNaughton, "Empirical failure analysis and validation of fault models in CMOS," in *Proceedings of the IEEE International Test Conference*, pp. 938–947, Sept. 1990.

[4] P. C. Maxwell, R. C. Aitken, V. Johansen, and I. Chiang, "The effect of different test sets on quality level prediction: When is 80% better than 90%?," in *Proceedings of the IEEE International Test Conference*, pp. 358–364, Oct. 1991.

[5] F. Ferguson and J. Shen, "Extraction and simulation of realistic CMOS faults using inductive fault analysis," in *Proceedings of the IEEE International Test Conference*, pp. 475–484, Sept. 1988.

[6] K. C. Y. Mei, "Bridging and stuck-at faults," *IEEE Transactions on Computers*, pp. 720–727, July 1974.

[7] M. Abramovici and P. Menon, "A practical approach to fault simulation and test generation for bridging faults," *IEEE Transactions on Computers*, pp. 658–663, Sept. 1985.

[8] J. M. Acken and S. D. Millman, "Accurate modeling and simulation of bridging faults," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 17.4.1–17.4.4, 1991.

[9] J. M. Acken and S. D. Millman, "Fault model evolution for diagnosis: Accuracy vs. precision," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 13.4.1–13.4.4, 1992.

[10] G. S. Greenstein and J. H. Patel, "E-PROOFS: A CMOS bridging fault simulator," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 268–271, Nov. 1992.

[11] B. Chess and T. Larrabee, "Bridge fault simulation strategies for CMOS integrated circuits," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 458–462, June 1993.

[12] C. Timoc, M. Buehler, T. Griswold, C. Pina, F. Scott, and L. Hess, "Logical models of physical failures," in *Proceedings of the IEEE International Test Conference*, pp. 546–553, Oct. 1983.

[13] R. E. Bryant, "COSMOS: A compiled simulator for MOS circuits," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 9–16, June 1987.

[14] D. G. Saab, R. B. Mueller-Thuns, D. Blaauw, J. A. Abraham, and J. T. Rahmeh, "Champ: Concurrent hierarchical and multilevel program for simulation of VLSI circuits," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 246–249, Nov. 1988.

[15] T. Lee and I. Hajj, "A switch-level matrix approach to transistor-level fault simulation," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 554–557, Nov. 1991.

[16] W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," *Ph.D. dissertation, University of California, Berkeley*, 1975.

[17] J. M. Acken, "Deriving accurate fault models," *Ph.D. dissertation, Stanford University*, 1988.

[18] S. Millman and S. J. Garvey, "An accurate bridging fault test pattern generator," in *Proceedings of the IEEE International Test Conference*, pp. 411–418, Oct. 1991.

[19] J. Ferguson and T. Larrabee, "Test pattern generation for realistic bridge faults in CMOS IC's," in *Proceedings of the IEEE International Test Conference*, pp. 492–499, Oct. 1991.

[20] T. Niermann, W.-T. Cheng, and J. Patel, "PROOFS: A fast, memory-efficient sequential circuit fault simulator," *IEEE Transactions on Computer-Aided Design*, pp. 198–207, Feb. 1992.

[21] F. Brglez and H. Fujiwara, "A neutral netlist of ten combinational benchmark circuits and a target translator in FORTRAN," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, Sept. 1985.

[22] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1929–1934, Sept. 1989.

[23] P. B. Cohen, "Using the tools," in *Octtools 5.1: Part 1: User Guide* (A. Casotto, ed.), Berkeley: Electronics Research Laboratory, University of California, 1991.

[24] A. Jee, "Carafe: An inductive fault analysis tool for CMOS VLSI circuits." Technical Report UCSC-CRL-91-24, University of California at Santa Cruz, Computer Engineering Department, Feb. 1990.

[25] T. Niermann and J. Patel, "Hitec: A test generation package for sequential circuits," *European Design Automation Conference*, pp. 214–218, Feb. 1991.