Bridge Fault Simulation Strategies for CMOS Integrated Circuits

Brian Chess
Tracy Larrabee *

Computer Engineering Board of Studies
University of California, Santa Cruz 95064

Abstract

After introducing the Primitive Bridge Function*, a characteristic function describing the behavior of bridged components, we present a theorem for detecting feedback bridge faults. We discuss two different methods of bridge fault simulation, one of which is new, and present experimental results relating the relative efficiency of the two methods. We conclude that the new simulation method,* Wire Memory *bridge fault simulation, is more efficient—especially for larger circuits.*

I. Introduction

Obtaining low IC defect levels requires that the ICs' tests have very high levels of fault coverage. Defect simulation experiments have shown that the vast majority of all local defects in MOS technologies cause changes in the circuit description that result in bridges and breaks [8, 13]. Most MOS fabrication technologies have more extra-conductor defects than extra-insulator defects, which makes accurate detection of bridge faults even more crucial.

We use the *Carafe* fault extractor to extract realistic bridge faults in CMOS circuits [10]. In the rest of the paper we will refer to bridge faults, which will always mean Carafe-extracted realistic bridge faults on wires between gates.

A *faulted* circuit is an isomorphic copy of an associated *fault-free* circuit except for the introduction of a change known as a fault. Some input combinations, when applied both to the fault-free circuit and to the faulted circuit, will produce identical outputs: in this case we say the input combination does not produce a *visible discrepancy*, and it is not a *test* for the introduced fault. If there is no input combination that produces a visible discrepancy, the introduced fault, considered in isolation, can never change the logic function of the circuit: in this case we say the fault is undetectable (or that it is a redundant fault). We now consider two different models for how circuits can become faulted.

In the *stuck-at* fault model, we assume that a circuit becomes faulted because a wire has lost its ability to switch values; the wire is *stuck-high* or *stuck-low*. The faulted circuit is identical to the fault-free circuit except for this one wire with the unvarying value. If this wire has a permanent value of 0 in the faulted circuit, and an input set causes the corresponding wire in the fault-free circuit to take on the value 0, the input set will create no visible discrepancy. If the wire takes on the value 1 in the fault-free circuit, we say that a discrepancy is introduced, and the fault is *stimulated*, but we still don't know if this introduced discrepancy will be visible at a circuit output. If the input set produces a visible discrepancy, we say that the introduced discrepancy has been *propagated* to a circuit output. A successful test must stimulate and propagate a fault.

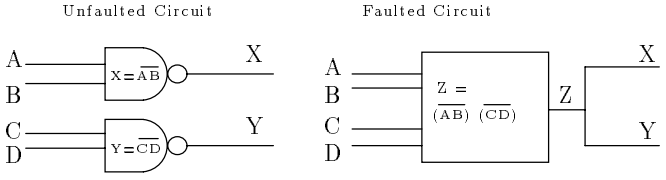

Figure 1: A Primitive Bridge Function derived from the wired-AND model

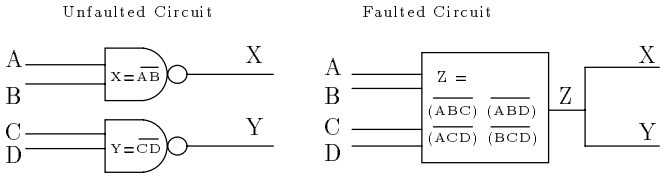

Figure 2: A Primitive Bridge Function derived from analog circuit analysis

In the *bridge* fault model, we assume that a circuit becomes faulted because two wires that are not connected in the fault-free circuit are connected in the faulted circuit. The bridge fault transforms the two gates for which the bridged wires are outputs into a single fault block in the faulted circuit. Figures 1 and 2 show how a bridge fault between the outputs of two NAND gates creates a fault block in the faulted circuit. The function of the new fault block is dependent on the behavior of the bridged components in the chosen technology. We refer to the characteristic function of the fault block as the *Primitive Bridge Function* or PBF. Figures 1 and 2 show two possible PBFs for the introduced fault block. The PBF in Figure 1 is the one used if the

*Work supported by NSF grant MIP-9158490.

technology in question follows the wired-AND model, and Figure 2 shows a PBF derived from circuit analysis of the CMOS MCNC standard cell components.

Many people have chosen the PBF to be the logic-AND or logic-OR of the two fault-free gate outputs [1, 9, 12]. These models are inaccurate for CMOS circuits, where the PBF will vary depending on the size, function, and technology of the bridged components. The voting model or an analog circuit analysis might be used to determine a more accurate Primitive Bridge Function [2, 3, 4]. It is possible that the inputs being driven by the bridged node may interpret the bridged voltage as different logic values because of different logic thresholds at the inputs [5]. This paper explores the case where this factor does not contribute to the final outcome and the PBF can be determined by circuit simulating only two components.

A combinational test for a bridge fault shares the same basic characteristics as the test for a stuck-at fault. To introduce a discrepancy, the output of the fault block must be different from one of the gate outputs in the fault-free circuit. To propagate the fault, we must produce at least one path of wires with discrepancies between their fault-free and faulted values from the fault block to a circuit output. The process of stimulating and propagating the discrepancy is complicated by the possibility of the bridge fault creating feedback.

If a bridge fault creates a feedback loop, a formerly stable combinational circuit may oscillate or take on sequential characteristics that mask the detection of the fault. It is possible to detect some feedback bridge faults that create sequential behavior with sequences of test vectors [12], but we have found that, as reported by Abramovici and Menon [1], the vast majority of feedback bridge faults can be detected with a single combinational test.

Previously, explicit bridge fault simulation was thought to be unwieldy because of the number of feasible bridge faults and the complexity of the bridge fault model. While the number of possible bridge faults is $O(n^2)$ where n is the number of nodes in the circuit, the number of realistic bridge faults is a much more manageable $O(n)$. Also, the number of different PBFs needed to analyze the fault blocks is not prohibitive because only one PBF is needed for each combination of bridged components. We will compare numbers of stuck-at faults, realistic bridge faults, and PBFs in a later section.

II. Foundations of Bridge Fault Simulation

In the process of simulating an input pattern against a given fault, there are some situations that will cause us to reject the pattern because we cannot *guarantee* that the combinational test will detect the bridge fault. For bridge faults without feedback, these situations are few: the fault does not create a discrepancy, or the discrepancy cannot be propagated. For bridge faults with feedback, the combinational test may be rejected for the same reasons that would cause a non-feedback bridge fault to be rejected, but there are 2 additional situations to consider.

A circuit with a feedback bridge fault will have a state (or will be oscillating between two unstable states). In order to combinationally detect the fault, we must have an input pattern that will detect the fault regardless of any previous state and leave the circuit in a stable state after application of the test. We might get a pattern that would detect the fault if the previous state were one of two possible values, but not the other value: we would reject such a pattern because we cannot guarantee detection. Similarly, because the tester may not detect the discrepancy, we would reject an input pattern that would cause the faulted circuit to oscillate.

When discussing feedback bridge faults, it is useful to refer to the two bridged wires by their locations in the circuit. Take any path that goes from a circuit input to a circuit output and contains the two bridged wires. The *back* wire is the wire closest to the circuit inputs on this path, and the *front* wire is the other bridged wire.

Before presenting the theorem upon which our bridge fault ATPG system is based, we will present two lemmas.

Lemma 1. A test can cause oscillation only if and only if a discrepancy on the back wire feeds back to one or more inputs to the fault block and causes the output of the fault block to change.

First let us show that oscillation will occur: Consider a feedback bridge fault where a discrepancy on the back wire is propagated through the front wire. When the value on the front changes, the value on the bridge will change because the bridged wires will no longer be driven to different values. Since there is no longer a discrepancy on the back wire, the front wire will soon return to its fault-free value, and the bridge will once again produce a discrepancy. This cycle will continue; the bridge will oscillate.

Now let us show that oscillation cannot occur unless there is a discrepancy on the back wire: By contradiction, assume that there is a test for a feedback bridge fault that introduces a discrepancy on the front wire and also causes oscillation.

First, assume that the value on the bridge before the application of the test is also the unfaulted value on the back wire. This means that none of the inputs to the fault block are different from the values in the unfaulted circuit, therefore the PBF will continue to assign a discrepancy to the front wire and no oscillation will occur.

Now assume that the value on the bridge before the application of the test is different than the unfaulted value on the back wire. In order to cause oscillation, this value must propagate to the inputs to the fault block and cause the PBF to change the value on the bridge. If the value on the bridge changes, the value on the bridge will now be the same as the unfaulted value on the back wire. As shown above, if the value on the bridge is equal to the unfaulted value of the back wire, the bridge will not oscillate.

If a discrepancy appears on the front wire, the only feedback path introduced by the bridge is forced to be inactive. Without feedback, oscillation cannot occur. □

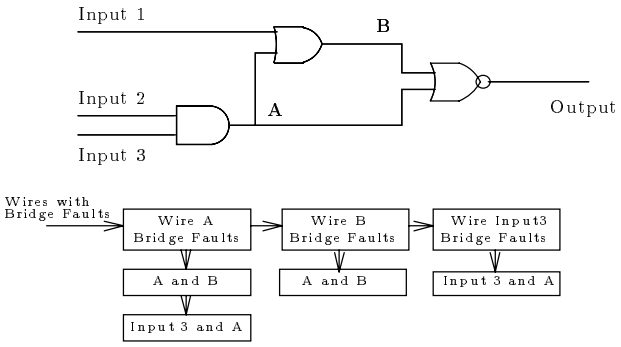

Figure 3: Data Structure for Wire Associative Bridge Fault Simulation

Lemma 1 implies that if there is no fanout between the back and front wires, it is impossible to create a reliable test with the discrepancy placed on the back wire.

Lemma 2. A test can be invalidated by sequential behavior if and only if the output of the fault block is dependent on the value on the back wire.

First let us show how sequential behavior can invalidate a test: Consider a feedback bridge fault before the application of a test. Since we are applying a single combinational test, the value on the bridge before the application of the test is unknown, and therefore the value on the back wire is unknown. Because the value of the fault block is dependent on the value of the back wire, the test cannot be guaranteed.

Now let us show that sequential behavior cannot occur unless the output of the fault block is dependent on the value of the back wire: If the output of the fault block is not dependent on the only wire representing any possible previous state (the back wire), it is dependent only on combinationally set values. □

Test Guarantee Theorem. A test for a feedback bridge fault cannot sensitize the output of the fault block to the back wire.

As a consequence of Lemma 1 and Lemma 2, neither oscillation nor sequential behavior can invalidate a test unless the output of the fault block is sensitized to the back wire. □

III. Two Bridge Fault Simulation Methods

We implemented two methods of bridge fault simulation and used each of them in the Nemesis ATPG system [7, 11]. Nemesis uses parallel pattern, single fault propagation (PPSFP) simulation [14] for pre-simulation and single pattern, single fault propagation (SPSFP) simulation after algorithmic vector generation. We place great emphasis on incorporating methods of bridge fault simulation into the PPSFP model, but we will report on parallel and single pattern simulation.

While stuck-at and bridge fault simulators will have different stimulation routines, they can share propagation routines. Pre-processing to ensure that the test does not create indeterminate results will only be necessary for bridge fault simulation.

A. Wire Associative Simulation

The first method of bridge fault simulation we will describe is a generalization of the wired-AND/wired-OR method originally presented by Abramovici and Menon [1]. We call this the *Wire Associative* method of bridge fault simulation because bridge faults are associated with the wires they fault. Figure 3 shows a diagram of the data structures used, and a pseudo-code description of Wire Associative simulation follows:

```
Simulate the fault-free circuit with test vector T
foreach wire (W) involved in a bridge fault
  if T detects a stuck-at fault on W
    for each bridge fault (BF) associated with W
      if the PBF for BF places a discrepancy on W
        Accept test T: BF is detected
      else
        test T does not detect BF from wire W
```

When we say that the PBF places a discrepancy on W, we mean that if the fault is a feedback bridge fault, we use the Test Guarantee Theorem to assure that the test cannot be invalidated be by oscillation or sequential behavior.

There are three major improvements that can be made to the basic Wire Associative algorithm:

1. Detecting a stuck-at fault is more time consuming than using the PBF to determine the output of the fault block. It is important to make sure at least one bridge fault will produce a discrepancy on the wire before the wire is simulated. If this optimization is not included, we will waste a large amount of time once we have covered most of the faults. This optimization is effective even if our technology allows us to use the wired-AND or wired-OR model.
2. If a bridge fault is not detectable by a test, the fault will be encountered twice as the list of wires with faults is traversed, once for each of the faulted wires. This means that the fault block could be examined twice with the same set of inputs. If the time spent satisfying the PBF is not trivial, it is a good idea to save the output of the fault block after it is simulated for the first wire so the value can be used for the second wire. This optimization is not important if our technology allows us to use the wired-AND or wired-OR model.
3. Following Lemma 1, no attempt should be made to place a fault on the back wire of a bridge fault if the discrepancy must travel through the front wire. This optimization is effective regardless of the technology.

B. Wire Memory Simulation

Our second method of bridge fault simulation does not associate bridge faults with wires; instead, wires are allowed

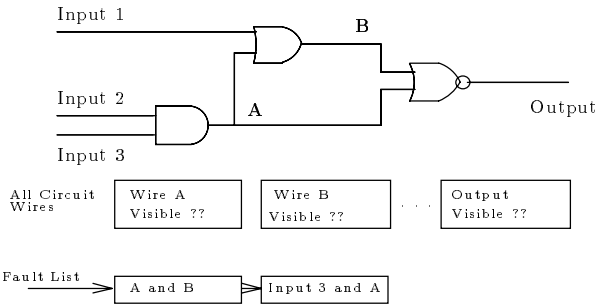

Figure 4: Data Structure for Wire Memory Bridge Fault Simulation

to "remember" if a fault can be propagated to a primary output. Accordingly, we call this the *Wire Memory* method of bridge fault simulation. After an attempt to propagate a discrepancy from a wire is made, a field in the wire's data structure is set to reflect the success of the propagation for this input set. If a fault further down the fault list introduces a discrepancy onto the same wire, it can immediately be determined whether or not the discrepancy can be propagated. Figure 4 shows the data structures required for Wire Memory fault simulation, and the pseudo-code for the Wire Memory method follows:

Simulate the fault-free circuit with test vector T
foreach bridge fault (BF)
  if the PBF for BF places a discrepancy on a wire (W)
    if a previous simulation of a fault on W can be used
      use previous simulation data
    else
      simulate fault on W introduced by fault block
    if the fault introduced by the fault block is detectable
      accept test T for this fault: BF is detected
    record results of simulation of fault on W for future use

The Wire Memory method of simulation offers an advantage that is not available to the Wire Associative method—in parallel simulation, faults can be propagated from both of the wires involved in the bridge. The reason we can do this is that the fault block does not introduce a discrepancy on both of the wires for any input pattern: the discrepancy is always on one wire or the other. This means that each bit-slice in the pair of faulted and fault-free wire values may represent a discrepancy on one wire or the other, but not both. A wire is placed on the simulation event queue if its faulted and fault-free values differ in any bit-position—regardless of whether the difference represents a value propagated from the fault on the first wire or the second wire. If the two bridged wires share a significant number of downstream components, the number of individual component simulations can be greatly reduced.

As with the Wire Associative method, no attempt should be made to propagate a fault on the back wire of a bridge if the discrepancy must travel through the front wire in the bridge.

IV. Experimental Results

Table 1 shows the number of stuck-at faults, the number of Carafe-extracted bridge faults, and the number of Primitive Bridge Functions associated with bridge faults for the MCNC layouts of the ten ISCAS-85 benchmark circuits [6].

Table 2 breaks bridge faults into 2 major categories: bridge faults that are capable of producing feedback and bridge faults that are not capable of producing feedback. Feedback bridge faults are subdivided into two groups. If, for a particular fault, every path from the back wire to a primary output goes through the front wire, the fault is a *feedback with no fanout fault* (FNF). If some but not all of the paths from the back wire to a primary output go through the front wire, the fault is a *feedback with fanout fault* (FWF). Our Lemma 1 can be directly applied to each FNF bridge fault.

Tables 3 and 4 compare the Wire Associative and Wire Memory algorithms for PPSFP and SPSFP simulation. Any optimization performed by the Wire Memory method that can possibly be applied to advantage for the Wire Associative method is included in the Wire Associative implementation so that the comparison is fair. Both for the PPSFP simulation and the SPSFP simulation the Wire Memory method is almost always faster than the Wire Associative method, and the improvement becomes more striking as the size of the circuits increase. Both methods can run on a machine with 16 megabytes of RAM.

We have had greater success with the Wire Memory method for a number of reasons. The ability to abort simulations, which can only be done in the Wire Memory method, saves a great deal of time. Also, the data structures needed for the Wire Memory method were easily integrated into a system (such as Nemesis) that treats many different types of faults (such as bridge, I_{DDQ}, and stuck-at) in a similar fashion. Data structure manipulation in the Wire Associative method is more complex because each fault appears twice (once for each wire that may carry a discrepancy). We anticipate this effect will become even more pronounced when we begin to take the logic threshold levels of components downstream from a bridge into account; this is because simulation of such a bridge faults may require propagation of multiple stuck-at faults.

Circuit	Stuck-At	Bridge	PBFs
C0432	564	1546	118
C0499	786	2747	36
C0880	966	3227	156
C1355	1882	4356	58
C1908	1246	4669	117
C2670	2237	13589	191
C3540	3185	16316	254
C5315	4865	40143	247
C6288	8748	21475	33
C7552	6291	53439	253

Table 1: Number of faults and PBFs for each circuit

Circuit	Feedback	FWF	FNF	No Feedback
C0432	880	828	52	666
C0499	1170	1134	36	1577
C0880	633	579	54	2594
C1355	1786	1754	32	2570
C1908	1827	1728	99	2842
C2670	1394	1244	150	12195
C3540	3607	3433	174	12709
C5315	3511	3203	308	36632
C6288	10590	10389	201	10885
C7552	4446	4039	407	48993

Table 2: Breakdown of circuit bridge fault statistics

		Time in Seconds	
Circuit	Faults covered	Wire Assoc.	Wire Mem.
C0432	1504	4.1	1.8
C0499	2740	1.8	1.7
C0880	3194	2.1	1.6
C1355	4163	10.3	9.7
C1908	4624	15.3	13.7
C2670	13090	17.5	21.2
C3540	16201	42.3	28.5
C5315	40019	27.6	18.5
C6288	20908	142.5	109.7
C7552	52662	435.2	287.2

Table 3: Nemesis Random Parallel Simulation

		Time in Seconds	
Circuit	Faults covered	Wire Assoc.	Wire Mem.
C0432	1504	15.8	10.8
C0499	2740	12.6	9.2
C0880	3194	21.1	16.9
C1355	4163	213.3	125.4
C1908	4624	130.3	96.0
C2670	13090	210.4	165.7
C3540	16201	323.0	237.9
C5315	40019	290.2	219.3
C6288	20908	1143.6	661.9
C7552	52662	4603.1	3528.2

Table 4: Nemesis Random Single Vector Simulation

V. Conclusions

We introduced the Primitive Bridge Function, a characteristic function describing the behavior of bridged components. We have also presented a theorem for tests that detect bridge faults After describing and presenting experimental results relating the relative efficiency of two bridge fault simulation strategies, we conclude that the new Wire Memory strategy is a significant improvement on previous bridge fault simulation strategies.

Acknowledgements

We thank Dr. Krzysztof Kozminski and MCNC for providing standard-cell layouts of the ISCAS benchmark circuits. We thank Haluk Konuk for useful technical discussions. The experiments were run on a Digital Equipment Corporation Decstation 5000/240 given us as part of Digital Equipment Corporation's external research grant.

References

[1] M. Abramovici and P. R. Menon. A practical approach to fault simulation and test generation for bridging faults. *IEEE Transactions on Computers*, C-34:658–663, 1985.

[2] J. M. Acken. Testing for bridging faults (shorts) in cmos circuits. In *Proceedings of Design Automation Conference*, pages 717–718, 1983.

[3] J. M. Acken. *Deriving Accurate Fault Models*. PhD thesis, Stanford University, Department of Electrical Engineering, September 1988.

[4] J. M. Acken and S. D. Millman. Accurate modeling and simulation of bridging faults. In *Proceedings of the Custom Integrated Circuits Conference*, pages 17.4.1–17.4.4, 1991.

[5] J. M. Acken and S. D. Millman. Fault model evolution for diagnosis: Accuracy vs precision. In *Proceedings of the Custom Integrated Circuits Conference*, 1992.

[6] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinatorial benchmark circuits and a target translator in fortran. In *International Symposium on Circuits and Systems*. IEEE, June 1985.

[7] F. J. Ferguson and T. Larrabee. Test pattern generation for realistic bridge faults in CMOS ICs. In *Proceedings of International Test Conference*, pages 492–499. IEEE, 1991.

[8] F. J. Ferguson and J. P. Shen. A CMOS fault extractor for inductive fault analysis. *IEEE Transactions on Computer-Aided Design*, 7(11):1181–1194, November 1988.

[9] A. Friedman. Diagnosis of short-circuit faults in combinational circuits. *IEEE Transactions on Computers*, pages 750–752, July 1974.

[10] Alvin Jee and F. Joel Ferguson. Carafe: An inductive fault analysis tool for cmos vlsi circuits. In *Proceedings of the IEEE VLSI Test Symposium*, page in press, 1993.

[11] T. Larrabee. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, pages 6–22, January 1992.

[12] K.C.Y. Mei. Bridging and stuck-at faults. *IEEE Transactions on Computers*, C-23(7):720–727, July 1974.

[13] J.P. Shen, W. Maly, and F.J. Ferguson. Inductive fault analysis of MOS integrated circuits. *IEEE Design and Test of Computers*, 2(6):13–26, December 1985.

[14] J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. McCarthy. Fault simulation for structured VLSI. *VLSI Design*, VI:20–32, 1985.