Test Pattern Generation for Realistic Bridge Faults in CMOS ICs

F. Joel Ferguson and Tracy Larrabee

UCSC-CRL-91-30
August 23, 1991

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

Two approaches have been used to balance the cost of generating effective tests for ICs and the need to increase the ICs' quality level. The first approach favors using high-level fault models to reduce test generation costs at the expense of test quality, and the second approach favors the use of low-level, technology-specific fault models to increase defect coverage but lead to unacceptably high test generation costs. In this report we (1) present the results of simulations of complete single stuck-at test sets against a low-level model of bridge defects showing that an unacceptably high percentage of such defects are not detected by the complete stuck-at test sets; (2) show how low-level bridge fault models can be incorporated into high-level test generation; and (3) describe our system for generating effective tests for bridge faults and report on its performance.

1. Motivation

The goal of a production testing system is to cost effectively minimize the number of bad parts that pass the tests. The number of bad parts that pass the tests are measured in *defective parts per million*, or DPM. McCluskey and Buelow have shown a simple relationship between the defect coverage of a test set, the yield of the manufacturing process, and the quality level of the circuits that passed the test set assuming that defects are independently distributed on the chip [MB88]. Table 3 from McCluskey and Buelow's paper shows that to obtain the relatively modest quality level of 200 DPM in a circuit with a 90% yield requires that the test set detect 99.8% of the manufacturing defects. If the yield is 75%, the test set must detect 99.93% of the defects.

Recent evidence shows that many defects are not detected by test sets that detect all single stuck-at faults (hereafter called *complete SSF test sets*) in some circuits [FS88, SM90, PRM90]. The studies reported by Ferguson and Shen, and Storey and Maly consisted of defect simulations to determine which circuit-level defects were probable, followed by a fault simulation to find the defect coverage [FS88, SM90]. The study by Pancholy, et al., involved fabricating easily diagnosable integrated circuits, testing them with a diagnostic test set, and applying a complete SSF test set to the faulty ICs. The complete SSF test set detected only 98.74% of the faulty blocks on the ICs fabricated by Pancholy, et al.

Our approach for increasing the quality (decreasing the defect level) of shipped ICs is to determine which faults are likely to occur, then find which of these are not detected by the SSF test set, and generate tests that detect the undetected faults. Likely faults are determined by simulating the results of known defect mechanisms, such as dust and other contaminants in the fabrication process, to the physical layout of the circuit. This is done as an automated defect simulation on the circuit layout. We refer to the faults that result in the defect simulation as *realistic faults* since they can be realized by known defect causing mechanisms.

The output of earlier defect simulators were unsuitable for automatic test pattern generation (ATPG), and the time for fault simulation was too great for the simulator to be combined with a production ATPG system. We avoid these disadvantages by first identifying realistic faults, which are generally presented as a change in the netlist of the circuit, and then translating these into changes in the logical function of the circuit. This allows fault simulation and test generation for the circuit with a much better performance than if fault simulation and ATPG were done at the switch or transistor level.

2. The Scope of this Report

The discussion in this report is restricted to the domain of fault modeling and ATPG for circuits designed using standard cells. The approach presented here can be extended to apply to gate arrays and other ASIC technologies, but is more easily implemented for standard cell designs. We partition the realistic faults into three categories, and the tests are generated for each fault based on which category it is in. The three categories are bridges and breaks within the cell, bridges in the interconnect, and breaks in the interconnect. In future work we will consider less common faults, such as, bridges between cells and interconnect, and bridges between adjacent cells.

The behavior of a cell with a bridge or break fault can be determined using a circuit-level simulator such as Spice. The computation costs of circuit-level simulation can be amortized over the life of the standard cell library in the same way that the costs of characterizing the library are amortized. Also the relatively small size of most cells makes circuit-level simulation feasible. The faults in a large cell can be characterized by partitioning the cell into smaller cells and characterizing them independently.

The result of the circuit simulation for a cell would be a list of input vectors for each cell that sensitize the cell's inputs and the resulting faulty behavior. The input cubes for sensitizing the cell and the location of the error could be used by a logic level ATPG system. For simple cells such as AND-OR-INV complex gates, it may be that SSF test sets are sufficient for detecting the realistic faults in the cell. We are currently characterizing the standard cells in the National Security Agency's CMOSn library. This will be reported on in the near future.

Breaks in signal lines cause the inputs to some of the receiving cells to be disconnected from the outputs of the driving cells, so that logic value of the driving cell's output does not affect the logic value at the receiving cells' inputs. For the sake of simplicity will assume in this paper that there is a single source for each signal. This means that there are no busses or signal lines with multiple tri-stateable drivers are considered. Signal line breaks may exhibit non-logical behavior but for most cases we can probably assume that the line "floats high" or "floats low". That is it behaves as a Stuck-at 0/1.

We can see from Figure /refline-break that a single defect can break a line that fans out to four gates so that it partitions the gates connected to that node in seven ways/footnoteIf

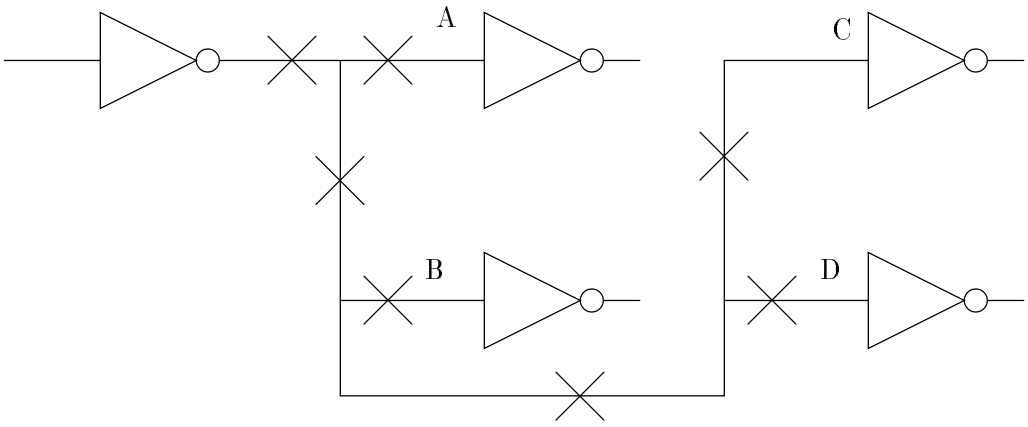


Figure 2.1: Potential Breaks in Circuit with Fan-out of Four.

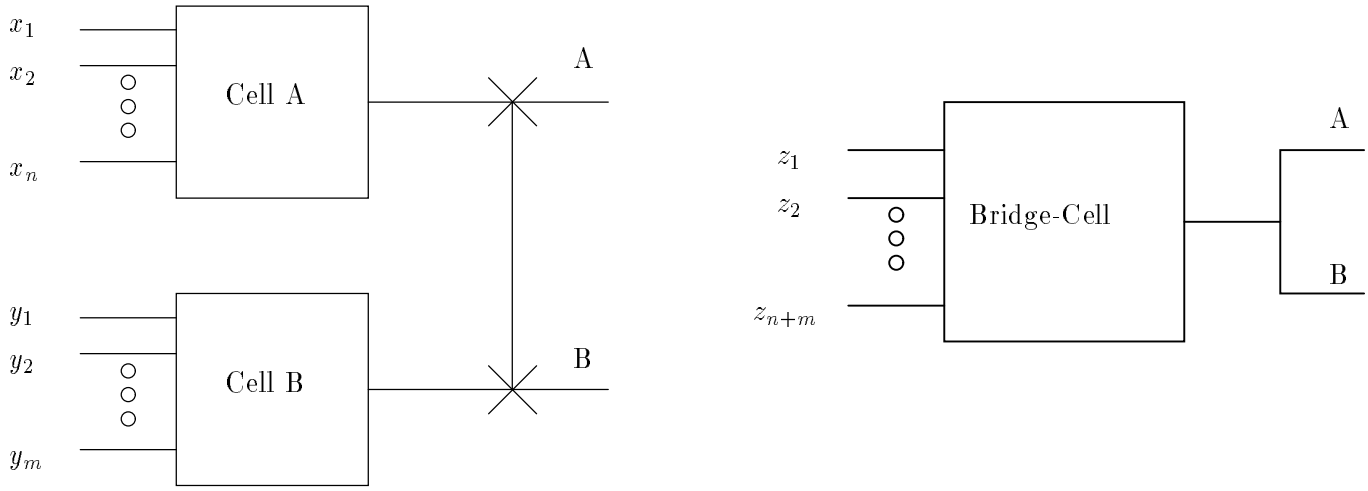


Figure 2.2: Modeling of Interconnect Bridge Faults.

breaks were allowed to occur in the area of the node that splits into parts, there are partitions of the circuit into three pieces.. Assuming that a nodes not connected to the output behave as stuck at 0 or 1, there are 14 single and multiple stuck-at faults that can be caused by a single break causing defect. There are 10 potential *single* line stuck-at faults for this node. If the topology of the circuit were different, for instance if gates A and D were swapped, the potential multiple-line stuck-at faults (MSFs) would change. A defect analysis tool such as VLASIC, FXT or Carafe can determine which multiple stuck-at faults may be caused by a single defect. This allows us to know which multiple stuck-at faults are likely in the circuit. This is much fewer than the set of all possible MSFs and may be feasible for ATPG.

Bridges and breaks in the standard cells and breaks in the interconnect were discussed in the previous paragraphs to show the general approach that we advocate for testing ASICs using standard cells, and to give an indication of how these realistic faults may be handled in the framework we are presenting. **The remainder of this report is concerned with the detection of bridge faults between signal lines connecting the standard cells only.**

Bridges in the interconnect layers are much more common than breaks in the interconnect layers for many fabrication processes [MTCC87]. As we see in the next section, the behavior of bridges cannot be modeled as a simple wired-or or wired-and. More generally a bridge between the outputs of two cells can be modeled as replacing the two cells with a single bridge-cell implementing the logic function of the bridged node as in Figure 2.2. In the next section we see that for some inputs to the bridge-cell there may be inputs for which the output logic value cannot be determined with certainty.

3. A System for Detecting Realistic Bridge Faults

We have divided the problem of generating test patterns for realistic bridge faults between signal lines into three parts:

1. Determine the realistic bridge faults. The program that performs this task is *Carafe*.
2. Find the change in local logical behavior for each fault. The program that performs this task is *Bridger*.
3. Generate tests detecting each change in local logical behavior. The program that performs this task is *Nemesis*.

The interfaces between these programs are shown in Figure 3.1.

3.1 Carafe

A version of Carafe (Circuit and ReAlistic Fault Extractor) is used to find the probable bridge faults between signal lines in the circuit [Jee90]. Carafe determines which bridge faults are possible by simulating the results of spot defects on the physical layout of the circuit. Defects are modeled as changes in the electrical behavior of small spots on the physical layers of the IC. Possible electrical changes for the top metal layer would be the presence of extra metal where there should be no metal, and the absence of metal where there should be metal. If the former happens and the spot of metal "touches" the conducting material of two circuit nodes, a bridge would occur between the nodes. Similarly a spot of missing metal could cause a break in a node. Such a defect in the physical layer could be caused by a particle of dust on the metal mask or in the photoresist used to shape the metal layer. For a more complete description of the determination of realistic faults see [FS88].

Carafe finds which bridges are possible by considering the range of probable spot defect sizes for each layer being considered and finding all nodes that are within that distance or closer to each other in that layer. These bridges are caused by defects that cause additional conducting material falling between these conductors. It also finds all nodes that can be bridged due to pinholes in the oxide layers. These nodes are on different layers such as between metal 1 and metal 2 and they must overlap with only a layer of oxide between them. Carafe takes as input the physical layout of the circuit to determine whether a bridge is likely and to determine how likely the bridge is.

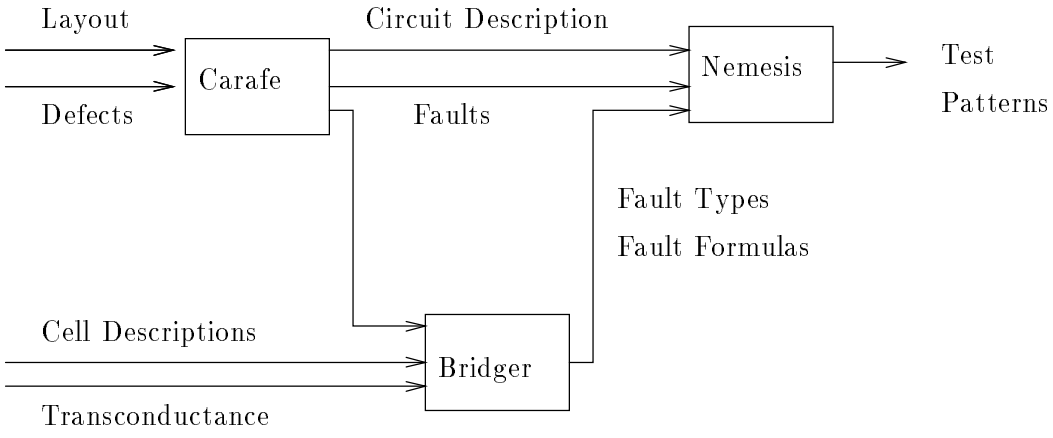


Figure 3.1: The Carafe-Nemesis Realistic Fault ATPG System

Test structures can be used to estimate the defect density of many types of spot defects on the IC [MTCC87]. Carafe uses this knowledge to estimate the probability that if a bridge fault occurs, how likely it is. One of Carafe's inputs is a file that describes the defect density and the distribution of defect sizes for each layer. Carafe determines the probability of each realistic bridge fault for each size defect. By weighing the results of the probability distribution function of defect sizes and summing them, Carafe determines the relative number of times each bridge fault should occur.

Normally Carafe flattens the layout to the transistor-level and describes the faults at the circuit-node level. A second version of Carafe was written that presents the circuit and faults at the logic-gate level for this research. This version of Carafe requires that the physical layout of the circuit be presented hierarchically with the two levels of hierarchy being the cells and the interconnect between the cells. In addition to providing the weighted fault list, this version of Carafe provides a gate level netlist for Nemesis and a list of fault types to Bridger. To evaluate Carafe's performance we ran it on the ISCAS-85 benchmark circuits collected by Brglez and Fujiwara [BF85]. The standard-cell layouts we used for the circuits were provided by MCNC. This version of Carafe's performance at extracting the realistic bridging faults of the MCNC cells on a Sparcstation 1+ is shown in Table 3.1.

Circuit	C432	C499	C880	C1355	C1908	C2670	C3540	C5315	C6288	C7552
faults	1546	2747	3227	4356	4669	13589	16316	40143	21475	53439
seconds	10.5	24.7	33.4	50.1	63.2	404.5	621.1	3326	921	5687

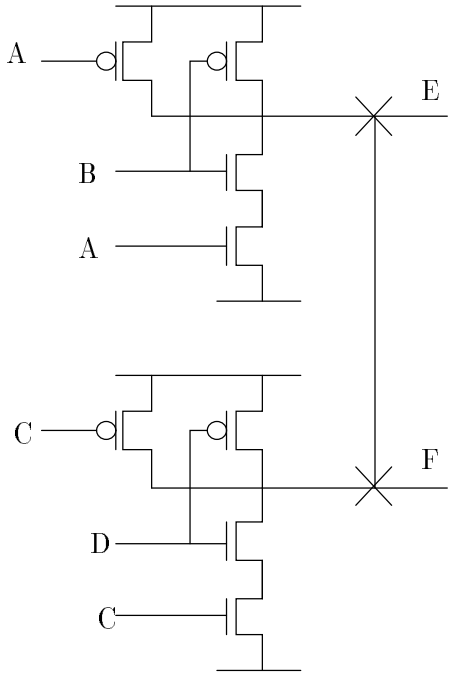
Table 3.1: Bridge fault extraction statistics for ISCAS-85 circuits using Carafe

3.2 Bridger

An example of how a simple wired–or or wired–and model does not accurately model a CMOS bridge is shown in Figure 3.2. The logic function caused by the bridge faults was obtained using a Spice simulation. The figure shows that two PMOS transistors in parallel are stronger than the NMOS transistors in series for NAND gates in the CMOS3 standard cell library [Hei88], but a single PMOS transistor is not. Note that the transistor strength model used in the COSMOS fault simulator cannot model this fault correctly by assigning any combination of strengths to the eight transistors.

Bridger uses the resistive model for conducting transistors that was suggested by Acken [Ack88] and later by Storey[SM90]. In experiments comparing the results of Spice simulations and Bridger, we found that if Bridger predicted the voltage at the bridged signals to be less than 2.0 Volts, Spice always predicted less than the logic threshold, and if Bridger predicted greater than 3.0 Volts, Spice predicted greater than the logic threshold. If Bridger's prediction was between 2.0 and 3.0 Volts, Spice would sometimes predict a different logical value than Bridger. We use voltage thresholds such that a logic 0 is anything between 0.0 and 2.0 Volts and a logic 1 is anything between 3.0 and 5.0 Volts: any more relaxed thresholds may lead to incorrect logic values.

For comparison with the strict threshold values detailed above, consider relaxed thresholds where voltages in the range 0.0 to 2.5 Volts is a logic 0 and voltages in the range 2.5 to 5.0 Volts is a logic 1. Some faults that are detectable with the relaxed threshold



Truth Table entries for E ≠ F

ABCD	EF	Wired-And	Wired-Or	Spice
0111	10	0	1	0
1011	10	0	1	0
0011	10	0	1	1
1101	01	0	1	0
1110	01	0	1	0
1100	01	0	1	1

Figure 3.2: Logic Function of Bridge Fault in CMOS3 NANDs using Spice.

are undetectable with the restricted thresholds. Table 3.2 shows the different numbers and percentages of an artificially constructed fault set (where each wire was bridged to the five wires directly following in the wirelist) that are detected by our complete SSF test set. The second and third rows of the table show the number of bridge faults detected by the complete SSF test using the different thresholds, and the last two lines show the percentage of detectable, non-feedback bridge faults (total bridge faults minus faults pruned due to feedback and faults declared untestable given the fault table for the given threshold) that are detected by the complete SSF test set.

Circuit	C499	C880	C1355	C1908	C2670	C3540	C5315	C6288	C7552
strict	902	1811	980	3846	5153	7464	10597	11846	15534
relaxed	902	1857	1537	4150	5517	7784	11102	11846	17078
% strict	99.9	98.6	66.8	96.3	95.9	98.2	98.2	100	96.0
% relaxed	99.9	99.6	99.9	99.7	98.0	99.5	99.3	100	99.2

Table 3.2: Randomly selected bridge faults covered by complete single stuck-at test set with strict and relaxed logic thresholds

3.3 Nemesis

Nemesis was originally an ATPG system for single stuck-at faults [Lar89]. It has been modified to take as input the AND-OR-INVERT cells in the MCNC standard cell library and to generate tests for the faults identified by Carafe and characterized by Bridger.

Nemesis generates a test pattern for a given fault in two steps: First, it constructs a formula representing all possible tests for the fault. Second, it applies a Boolean satisfiability algorithm to find a solution to the resulting formula. Any solution for the formula is a test that detects the fault [Lar92]. In order to generate tests for bridge faults, three routines were added to Nemesis and three routines were modified in Nemesis. The additional routines check for feedback when two wires are bridged, parse the list of fault types and faulted behaviors generated by Bridger, and parse the list of probable bridge faults generated by Carafe. The wirelist parser was modified to read wirelists generated by Carafe, and the simulator and formula extractor were modified so that they could simulate and extract formulas for bridge faults according to the constraints set by Bridger.

To evaluate Nemesis's generation of test patterns for bridge faults we have compared it with Nemesis's performance when generating test patterns for single stuck-at faults for the MCNC generated circuits. Table 3.3 shows the performance of the single stuck-at Nemesis standard-cell system. All times are taken with the program running on a Sparcstation 1. For comparison with the following tables, the results in Table 3.3 do not include global implications (learning). The "Untestable" row shows the number of SSFs that were proven untestable by Nemeis, the "Aborted" row shows those faults that no test pattern was generated for and were not proven untestable, and the "Seconds" row shows Nemesis's ATPG time.

Circuit	C432	499	C880	C1355	C1908	C2670	C3540	C5315	C6288	C7552
SSA Faults	826	974	1308	2430	1746	3176	4522	7022	11500	8506
Untestable	9	8	0	8	9	78	119	59	55	57
Aborted	7	0	0	0	0	14	0	2	0	60
Patterns	53	54	66	87	109	120	163	147	39	209
Seconds	21.0	5.9	23.1	36.6	37.2	257.3	125.9	109.4	263.8	466.0

Table 3.3: Nemesis test pattern generation single stuck-at statistics

Table 3.4 shows how extracted faults were pruned from the fault list before being sent to the ATPG part of the system. The row labeled "Total Faults" is the total number of faults generated by Carafe. Many gates, when bridged together, produce a voltage in the range that is not clearly a logic 0 or logic 1, 2.0 to 3.0 Volts, for all possible inputs except the inputs that produce a logic 1 on both gates or a logic 0 on both gates. An example of this would be a bridge fault between two inverters if the inverters' PMOS and NMOS transistors have equal strength. The row labeled "Removed – Untestable" shows the number of faults that are determined untestable by Bridger. These were discarded from the fault list before ATPG. The row labeled "Removed – Feedback" shows the number of bridges that introduced feedback to the system. These were removed from the fault list in this version of the system because such faults may introduce state to the circuit and are more difficult to test.

Table 3.5 shows the performance of the ATPG part of the system. The first row shows the number of bridging faults that are considered. The second line is the number of realistic bridge faults that were detected by the complete SSF test set presented earlier. The third line shows the number of realistic bridge faults that our Nemesis-based ATPG system detects. The last three lines give the number of bridge faults that are proven untestable,

Circuit	C432	C499	C880	C1355	C1908	C2670	C3540
Total Faults	1546	2747	3227	4356	4669	13,589	16,316
Removed – Untestable	727	1672	2106	1745	2094	9116	6295
Removed – Feedback	616	479	322	1154	953	915	2466
Sent to ATPG	203	596	799	1457	1622	3558	7555

Table 3.4: Pre-ATPG fault statistics

the number of faults that were aborted by the ATPG program, the number of tests that detect all bridging faults, and the time in seconds to completely process each circuit.

Circuit	C432	C499	C880	C1355	C1908	C2670	C3540
Faults	203	596	799	1457	1622	3558	7555
SSA covered	141	550	703	1025	1238	2291	6566
Covered	164	562	733	1058	1334	2474	6749
Untestable	38	34	65	238	286	1013	779
Aborted	1	0	1	161	2	71	27
Patterns	64	59	102	86	167	238	326
time (secs)	54.9	35.6	70.5	7500.8	286.9	2420.8	3101.0

Table 3.5: Nemesis test pattern generation bridge-fault statistics

The three largest circuits are not included in Tables 3.4 and 3.3 because the formulas to be satisfied for these circuits with bridge faults are too large for our workstations. Two things can be observed from Tables 3.3 through 3.5. The first is that this system detects many of the realistic bridging faults that are not detected by the SSA generated tests. The additional computation effort results in a considerable increase in realistic fault coverage. One also notices that the ATPG times for the bridging faults is much greater than one would expect. Why this is is not yet understood.

4. Conclusions and Future Work

We have presented an approach that allows one to make use of technology- and layout-specific realistic faults, which more accurately model faulty circuits generated from standard cell libraries, without sacrificing the advantages of fault simulation and ATPG at the logic-gate level. This is done by partitioning the set of faults into three main categories, intra-cell faults, bridges in the interconnect, and breaks in the interconnect, and applying the most appropriate fault modeling and test pattern generation techniques for each category.

Existing software have been modified to find the realistic bridge faults between interconnects, determine the resulting change in logical behavior caused by the bridge, and fault simulate and generate tests for them at the gate level. The exercise of integrating this software to generate tests for realistic faults at the logic gate level shows that it can be done, and pinpoints and helps quantify the problems with this approach. The biggest problem that has appeared is the number of faults. There are two to six times more bridge faults in the interconnect than there are single stuck-at faults. ATPG time increased by factors of two to twenty. Some of the slowdown is attributable to increased faults, but much is due to Nemesis's difficulty with the more complicated formulas. We are investigating modifications to Nemesis to make it more suited to dealing with formulas from large cells. The number of bridge faults we generate tests for can be reduced by considering only the most likely bridge faults. Carafe can provide this information.

A side-product of integrating the software is the verification that many detectable bridge faults are not detected by complete SSA test sets. For the MCNC standard-cell implementations, the percentage detected by complete SSA test sets ranges from 87% to 98%. We have also quantified some of the costs (for this *initial* implementation) of increasing the quality level by generating tests for the bridge faults that are not detected by the single stuck-at tests.

This research has exposed several areas for future research. The first is detecting the large number of untestable bridge faults. Most of these faults are untestable because the resulting output voltages from Bridger do not meet the logic thresholds to clearly be a logic 0 or 1. That is, the resistive model of the transistors predict an output voltage always being between 2 and 3 Volts when the bridged nodes are being driven to different logic values. Possible solutions are to employ IDDQ testing [Ack83], apply more accurate circuit simulation of faults, detect the bridge as a delay fault, or redesign the cells so that a discrepancy is guaranteed for at least one input combination for each cell. In Table 4.1 we show the results of our system generating IDDQ test patterns for same set of bridging faults that produced the results in Table 3.5. Test pattern generation time and the number of untestable faults is much less using IDDQ testing but this has to be traded off with the fact that the application of test vectors for IDDQ testing is slower than for logic level testing [Cra87].

We also plan on integrating the testing for breaks on the interconnection lines, and the testing for defects within the cells to Carafe and Nemesis. This would integrate all bridge faults and break faults into our testing framework except bridges between cells and bridges between a cell and adjacent signal lines.

Circuit	C432	C499	C880	C1355	C1908	C2670	C3540	C5315	C6288	C7552
Total Faults	1546	2747	3227	4356	4669	13589	16315	40143	21475	53439
Untestable	3	0	0	3	5	24	21	26	7	47
Aborted	0	0	0	0	1	2	0	0	0	4
Covered	1543	2747	3227	4353	4663	13563	16295	40117	21468	53388
Patterns	17	20	17	18	25	17	29	23	29	30
Time (secs)	1.1	2.2	1.6	4.5	38.7	35.8	14.3	23.0	12.9	115.0

Table 4.1: Test pattern generation IDDQ statistics (standard cell)

Acknowledgements

The authors thank Alvin Jee, Heather Trumbower, Rich McGowen, and David Staepelare for their work on our implementations. We also thank Dr. Krzysztof Kozminski and MCNC for providing us with standard-cell layouts of the ISCAS benchmark circuits, the Semiconductor Research Corporation for supporting this research under Contract 90-DJ-141, and The National Science Foundation for supporting this work under grants MIP-8907380 and MIP-9011254.

References

[Ack83] J.M. Acken. Testing for bridging faults (shorts) in CMOS circuits. *Proceedings of Design Automation Conference*, pages 717–718, 1983.

[Ack88] John M. Acken. *Deriving Accurate Fault Models*. PhD thesis, Stanford University, Department of Electrical Engineering, September 1988.

[BF85] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1985.

[Cra87] C. Crapuchettes. Testing CMOS idd on large devices. In *Proceedings of International Test Conference*, pages 310–315. IEEE, 1987.

[FS88] F. Joel Ferguson and John P. Shen. A CMOS fault extractor for inductive fault analysis. *IEEE Transactions on Computer-Aided Design*, 7(11):1181–1194, November 1988.

[Hei88] Dennis V. Heinbuch. *CMOS3 Cell Library*. Addison-Wesley Publishing Company, 1988.

[Jee90] Alvin 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, February 1990.

[Lar89] Tracy Larrabee. Efficient generation of test patterns using Boolean Difference. In *Proceedings of International Test Conference*, pages 795–801. IEEE, 1989.

[Lar92] Tracy Larrabee. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, January 1992.

[MB88] E.J. McCluskey and F. Buelow. IC quality and test transparancy. In *Proceedings of International Test Conference*, pages 295–301. IEEE, 1988.

[MTCC87] W. Maly, M.E. Thomas, J.D. Chinn, and D.M. Campbell. Double-bridge test structure for the evaluation of type, size and density of spot defects. Technical Report CMUCAD-87-2, Carnegie Mellon University, SRC-CMU Center for Computer-Aided Design, Dept. of ECE, February 1987.

[PRM90] Ashish Pancholy, Janusz Rajski, and Larry McNaughton. Empirical failure analysis and validation of fault models in CMOS VLSI. In *Proceedings of International Test Conference*, pages 938–947. IEEE, 1990.

[SM90] Thomas Storey and Wojciech Maly. CMOS bridging fault detection. In *Proceedings of International Test Conference*, pages 842–851. IEEE, 1990.