Efficient and Exact Diagnosis of Multiple Stuck-At Faults

J. Brandon Liu
University of Toronto
Department of ECE
Toronto, ON M5S 3G4
liuji@eecg.utoronto.ca

Andreas Veneris
University of Toronto
Department of ECE and CS
Toronto, ON M5S 3G4
veneris@eecg.utoronto.ca

Magdy S. Abadir
Motorola
7700 W. Parmer Austin, TX 78729
m.abadir@motorola.com

Abstract

This paper presents a simulation-based approach to multiple stuck-at fault diagnosis. The algorithm accepts a digital implementation corrupted with stuck-at faults and its simulatable gate-level netlist. It then iteratively identifies and fault models suspect locations one at a time until the set of faults modeled can functionally fully account for the faulty behavior. The method uses a theoretical result along with a number of heuristics to help avoid the exponential complexity inherent to the problem. Experiments on up to four multiple faults confirm its effectiveness and accuracy which scales well with increasing number of faults.

1. Introduction

The rising complexity of today's VLSI design and manufacture process necessitates the wide-spread use of *failure analysis* [3] to help tune the manufacture and/or design cycle, and improve the product yield. The quality of failure analysis is tightly coupled to this of **fault diagnosis**, that is, the identification of failing portions of the chip. Physically probing the faulty chip with electron beams is a slow and expensive process. Software-driven fault diagnosis alleviate cost by compiling a much more compact list of suspect lines from a logic gate-level implementation of the circuit, which a test engineer can then probe to determine the actual place and cause of the failure.

The stuck-at fault model on a line l in a logic boolean network dictates that l is permanently assigned to logic 0 or 1. It is widely used in the industry due to its simplicity and high fault coverage [1]. In multiple stuck-at fault diagnosis, more than one lines of the circuit have stuck-at faults and can cause functionally errors inter-dependently. Fault diagnosis in this setting is an inherently difficult problem because the space of suspect fault/line combination, if probed brute-force, grows exponentially with the number of errors: $error\ space\ = (3 \times \#\ of\ lines)^{\#\ of\ errors}$. Due to these reasons, the research community has shown significant interest in the development of efficient and accurate fault diagnosis algorithms [3] [7] [10] [11].

In this work, we propose a simulation-based incremental approach to multiple stuck-at fault diagnosis for combinational and full-scan sequential digital circuits. The input to the algorithm is a simulatable circuit implementation corrupted with multiple stuck-at faults (emulation of the faulty chip), its logic netlist and set of input test vectors $\mathcal{V}$ with erroneous primary output responses. During each iteration, the algorithm identifies a single suspect line in the netlist and injects an appropriate stuck-at fault on the line so to bring the function of the netlist "closer" to that of the faulty implementation. The algorithm terminates when the set of stuck-at faults can fully account for the functional errors we observe in the implementation.

This work differs from previous incremental diagnosis approaches [4] [5] in the methodology used to identify the fault locations and in the fact that explicit fault modeling occurs at every iteration. In subsection 3.2 we prove a theorem that stipulates a lower bound on the number of unmatched vectors each "valid" fault candidate *must* modify. This result, along with a number of different heuristics, allows efficient exclusion of candidates that cannot lead to an optimal solution. In practice, it helps avoid the exponential explosion of the problem. We also show that during diagnosis, it may be necessary to consider corrections that do not look optimal locally but have the potential to lead ultimately to an optimal solution. In addition, unlike previous approaches, the incremental framework presented here can handle *any* number of errors.

To exhibit the effectiveness of the proposed approach, we run experiments on designs corrupted with 1, 2, 3 and 4 stuck-at faults. The results exhibit excellent resolution, which scales well with increasing number of faults, and good efficiency as the method can examine thousands of candidates within seconds of CPU time. In addition, the algorithm can return nearly all possible sets of multiple stuck-at faults that explain the functional mismatch including the actual set of faults. This, to the best of our knowledge, makes this work the first *exact* multiple stuck-at fault diagnosis algorithm. Finally, the algorithm is simple to code and can tackle additional real-life industrial problems as it can be easily extended to diagnose erroneous implementations at different levels of the design cycle (RTL, logic, or physical).

In the Section that follows we give relevant definitions. Diagnosis and fault modeling are described in Section 3. Experiments on multiple stuck-at faults can be found in Section 4 and Section 5 contains the conclusions.

2 Preliminaries

In this work, we investigate netlists with logic `NOT`, `BUFFER`, `AND`, `NAND`, `OR` and `NOR` gates. Although the algorithm can handle `XOR` and `XNOR` gates, we do not consider them explicitly. Further, we assume that both the design netlist and the corrupted implementation are completely simulatable. This assumption can be relaxed as discussed in [8]. A line whose value changes during simulation under the presence of some fault(s) is called a *sensitized line* and a path consisting entirely of sensitized lines is called a *sensitized path*.

The behavior of the faulty chip is emulated by injecting stuck-at faults into the logic simulation of the fault-free netlist. Prior to the execution of the algorithm, we simulate all the ATOM vectors, which have 100% single stuck-at fault coverage [2], and a number of random input test vectors. They are split into create two separate sets, $\mathcal{V}_{err}$ and $\mathcal{V}_{corr}$. The former is used during diagnosis and both are used during verification. Each vector in $\mathcal{V}_{err}$ has at least one primary output (PO) erroneous. None of the vectors in $\mathcal{V}_{corr}$ has any erroneous primary output. The logic values of line l for each vector are saved correspondingly in two bit-lists: $\mathcal{V}^l_{err}$ and $\mathcal{V}^l_{corr}$. The i-th entry of the $\mathcal{V}^l_{err}$ ($\mathcal{V}^l_{corr}$) list contains the logic value of l when we simulate the i-th input test vector from $\mathcal{V}_{err}$ ($\mathcal{V}_{corr}$). These bit-lists are used and properly updated, as it will be explained in the next section. Deterministic test generation for the errors injected is not the subject of this work.

In this discussion, we refer to the wire that is under consideration as a *suspect* wire. Additionally, we call the combination of a suspect wire and a proposed fault model for this wire as a *candidate*. Using this terminology, the problem solved is formulated as follows. Given a logic netlist and a corrupted implementation for the netlist, we seek a set of n candidates that once applied on the netlist they make it behave identically to the corrupted implementation. Observe that a solution can be classified as either *actual* or *equivalent* [8]. We call any such solution a *valid* solution.

Finally, we use *path–trace* as the first step of diagnosis, which is a line marking procedure, similar to critical path tracing, developed for fault diagnosis by Venkataraman et al. [10]. For an erroneous vector v, path–trace starts from an *erroneous* primary output for v and traces backwards toward the primary inputs of the circuit, while marking lines of interest. Details and examples of this linear time algorithm can be found in [10]. Its importance to fault diagnosis stems from the fact that it always marks *at least* one line from every set of valid solutions [8].

3 Incremental Diagnosis Algorithm

The *input* to the method is an implementation corrupted with a number of stuck-at faults, its netlist, and a set of input test vectors $\mathcal{V}$. In the industrial setting, the implementation is the faulty chip itself. Due to our lack of access to physical chips with defects, we obtain the erroneous functional behavior by fault simulation. In our experiments, we simulate vectors from [2], which has 100% single stuck-at fault coverage, along with 2,000-5,000 random vectors to compile $\mathcal{V}^l_{corr}$ and $\mathcal{V}^l_{err}$ on every line l of the circuit. The *output* of the algorithm is a list of valid solutions. Considering the existence of equivalent solutions, it is evident that an accurate fault diagnosis algorithm should not only return results that localize to *some* faulty areas but also the *maximum* possible set of (equivalent) fault locations that explain the overall fault effects.

The algorithm works in an iterative manner. It bases its operational flow on a *decision tree*. Every iteration of this process is represented by a different level in the tree where different options (decisions) are available. At each iteration, it identifies a set of lines (suspect extraction) and devises a set of suitable fault model(s) (candidate selection) for these lines according to certain parameters described in the next two subsections. Each candidate is an option for additional iteration and extending the tree at the node. In most of the cases, each iteration reduces the number of erroneous primary outputs for *all* vectors in $\mathcal{V}_{err}$. It should be noted that the algorithm does not discard intermediate candidates that increase the number of EPO's. As discussed in [5], the number of erroneous primary output responses does not necessarily decrease with the number of faults present in the system. We develop heuristics, presented in subsection 3.2, that prevent the algorithm from deleting such valid corrections. Finally, a leaf node is found when a valid solution is identified (success leaf) or when a node exceeds the pre-defined maximum depth (failure leaf), that is, maximum number of faults we diagnose. A path from the root of the tree to a success leaf represents a valid solution.

Because the entire decision tree is to be traversed to avail *all* equivalent tuples, the global flow of the incremental process is not crucial for its success as it is in [9]. Either a breadth-first or a depth-first traversal will be sufficient. However, if only a partial tree is to be searched due to time/space constraints, a match-score guided search may reduce the cost of finding the first N valid solutions. This is not elaborated upon in this paper but a detailed algorithm can be found in [9]. In the next two subsections, we present the diagnosis steps.

3.1 Suspect Extraction

The objective of suspect extraction is to quickly reduce the fault space and eliminate lines that have no potential to lead to an optimal solution. This step is done with path-trace, which marks suspect lines in the circuit. It has been shown [8] that path-trace always marks at least one location from every set of locations where valid corrections exist. Therefore, we allow lines that have a high path-trace count to qualify for the subsequent candidate selection. During our experiments, we select the top 30-50% of these lines and visit them in the decreasing order of the path-trace counts. Our experiments indicate that lines with high counts often lead to valid solutions.

3.2 Candidate Selection

Although the number of suspect wires can be small, the validity of the fault model applied at each iteration can influence the overall performance since incorrect decisions may cause the underlying decision tree to explode exponentially. In this section we present a number of techniques that guide the fault modeling procedure and help avoid the potentially exponential explosion. The following theorem gives a necessary condition which all valid fault models *must* satisfy. As indicated in our experiments, this simple theorem provides a reliable guide to the candidate selection procedure.

Theorem 1: Let $l_1, l_2, \ldots, l_N$ be the set of lines where a set of valid fault models can be applied to fully account for the fault effect. Let V be a set of input test vectors with *failing primary output* responses. Define V_i to be the maximum subset of vectors from V such that its vectors produce an erroneous logic value on l_i (*i.e.,* excite the fault) and propagate this difference to some primary output(s). The minimum size of V_i that has the maximum cardinality among all sets $V_1 \ldots V_N$ is no less than $\frac{|V|}{N}$.

Proof: The theorem is an application of the pigeonhole principle of counting. By definition, each vector in V excites at least one fault to sensitize and/or co-sensitize a path(s) to some primary output(s). In other words, each vector in V is attributed to *at least* one V_i for some i, $1 \leq i \leq N$. Therefore, by the pigeonhole principle, the minimum size of the set V_i with the maximum cardinality among all such sets cannot be less than $\frac{|V|}{N}$.

The candidate selection algorithm proceeds as follows. Given a suspect location l that qualified, the algorithm *exhaustively* compiles a list of stuck-at-0 and stuck-at-1 fault models as in [4] [8]. From these, we keep the ones that satisfy both of the following screening tests after they are applied to suspect wire l to form a candidate:

Screening test on $\mathcal{V}^l_{err}$ vectors: Any qualifying fault model must complement at least $\frac{|\mathcal{V}^l_{err}|}{N}$ bits in $\mathcal{V}^l_{err}$ (*heuristic 1*). This screening test is a local application of Theorem 1 on the erroneous bit-list of l because if a line is to cause $\mathcal{M}$ erroneous primary outputs to be corrected (i.e. its logic value inverted), it has to invert at least $\mathcal{M}$ of its own logic values. With stuck-at faults, this test can be performed efficiently by comparing the suspected stuck-at value (i.e. logic 1 or 0) with those in the l_{err} array. The lower bound imposes a minimum number of bits in the array that have to differ from the stuck-at value.

Since the number of errors N is unknown, the lower bound stipulated by the theorem can not be determined. In implementation we follow an aggressive approach in applying this screening test. We first necessitate that a fault model complements a high percentage of local bit entries. We empirically set this limit to the initial value 70% and reduce it progressively when the algorithm returns with no candidates (*heuristic 2*). There are two reasons that justify such an approach. First, certain classes of faults are easier to excite than others. For instance, a stuck-at-1 fault at an output of an `AND` gate is easier to excite because this gate is expected to drive at most of the times the logic value 0. Second, Theorem 1 provides a conservative *lower bound* on the maximum cardinality of V_i. It does not account for vectors that simultaneously excite multiple faults and, consequently, can be attributed simultaneously to different V_i sets.

Screening Test on $\mathcal{V}^l_{corr}$: Any qualifying candidate may sensitize a small number of new paths to previously correct primary outputs (*heuristic 3*). We elaborate on the rationale behind this heuristic with an example.

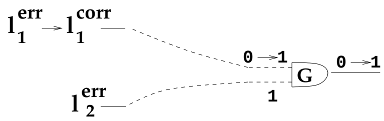

Figure 1. Example of Screening on $\mathcal{V}^l_{corr}$

Example 1: Fig. 1 depicts the situation where the effects of two faults on lines l_1 and l_2 sensitize two paths that merge in gate G with logic values 0 and 1 respectively. Also assume that this vector v produces correct primary output responses. It can be seen that in the correct implementation, l_1 produces a logic 1 in the first input of G and l_2 produces a logic 0 in the second input of that gate. Therefore, when a valid fault model is applied to l_1, under the presence of the second error, the logic value at the fan-out of G switches to 1. If a primary output is sensitized to the fan-out of G, v becomes erroneous. It remains erroneous until a valid fault model is applied to l_2 and the fan-out of G switches back to its correct logic value.

The above example suggests that sometimes we should accept candidates that sensitize a few new erroneous primary outputs. In our experiments, we require a correction to create no more than 3-8% new erroneous primary outputs

Table 1: Results on Stuck-At Faults (time in sec.)

ckt	ckt	1 fault		2 faults			3 faults			4 faults		
name	lines	# **sites**	time	# tuples	# **sites**	time	# tuples	# **sites**	time	# tuples	# **sites**	time
C499	1265	**1.1**	0.70	2.8	**3.4**	0.72	7.3	**5.8**	0.91	3.3	**5.9**	29.6
C880	910	**2.7**	0.18	6.9	**4.9**	0.21	7.1	**6.1**	0.59	9.4	**7.3**	4.16
C1355	1211	**3.7**	0.17	2.3	**3.0**	0.81	4.5	**5.5**	10.89	9.3	**7.3**	44.9
C1908	928	**1.0**	0.98	2.3	**3.2**	2.76	12.1	**6.5**	0.85	7.3	**6.8**	16.7
C2670	1459	**2.2**	0.69	6.2	**5.6**	0.44	4.5	**5.0**	2.33	31.3	**9.8**	13.8
C3540	2281	**1.2**	1.96	4.9	**4.5**	0.67	7.2	**6.1**	29.72	14.3	**8.3**	67.6
C5315	3698	**1.7**	1.57	2.9	**3.3**	3.06	6.4	**5.1**	15.93	4.1	**6.1**	276
C6288	6264	**1.1**	3.43	2.2	**2.8**	2.94	5.9	**4.7**	14.58	1.5	**4.5**	61.3
C7552	5251	**2.7**	1.61	5.2	**3.9**	8.16	13.4	**6.2**	6.32	9.3	**8.3**	175
S838	731	**2.2**	0.16	4.0	**3.3**	0.35	7.2	**5.0**	1.80	214.7	**15.3**	1.01
S953	1082	**1.9**	0.26	2.6	**3.4**	0.55	2.3	**2.3**	13.81	96.1	**17.2**	49.38
S1196	1246	**2.4**	0.36	2.7	**3.0**	0.86	6.8	**6.5**	3.47	5.5	**7.2**	16.89
S1494	1421	**2.9**	0.21	7.0	**5.3**	0.25	18.9	**9.0**	1.31	5.3	**6.3**	17.17
S9234	4640	**4.1**	0.73	31.3	**8.4**	0.36	116	**12.5**	2.89	17.3	**10.2**	33.87
Average		**2.2**			**4.1**			**6.2**			**8.6**	

on the average. Empirically, this number has been shown to be sufficiently large to allow valid corrections to qualify. We perform the screening test on $\mathcal{V}^l_{corr}$ efficiently through simulation on the $\mathcal{V}_{corr}$ bit-lists on the lines in the fan-out cone of l.

4 Experiments

We implemented the algorithm of Section 3 in C language and ran it on a SUN Ultra 5 workstation for ISCAS'85 and full-scan versions of the ISCAS'89 benchmark circuits corrupted by multiple stuck-at faults. The locations of the faults were selected at random. In the following paragraphs we present and discuss the results. All run–times are in seconds and do not include the initial random simulation step for $\mathcal{V}$ which is performed only once. However they include the time for updating the bit-lists at subsequent steps. The overall memory requirement for the algorithm is relatively small (less than 98MB).

In order to simulate a realistic diagnosis environment, we first optimize the ISCAS'85 and ISCAS'89 circuits for area using SIS [6] with $script.rugged$. Then we randomly inject 1, 2, 3 or 4 faults and run the algorithm exhaustively (i.e. traversing the decision tree fully) as we are interested in all possible sets of faulty locations (tuples) and associated stuck-at faults that explain the faulty behavior of the device. In other words, we require the algorithm to have an *exact* performance and return the maximum set of minimal-size equivalent stuck-at fault sites that justify the faulty behavior. This information is the most useful to a test engineer. We ran 10 experiments for each fault case and each circuit (i.e. forty experiments per circuit). The average performance of the algorithm can be found in Table 1.

The first two columns of Table 1 contain the circuit name and circuit lines, respectively. The next columns contain the information pertaining to each different fault case: the number of tuples (singles, pairs, triples, quadruples) returned that fully account for the faulty behavior (*i.e.*, *equivalent fault classes* [10]), the number of distinct sites in all these tuples the test engineer needs to probe (explained below), and the average run-time required to discover one tuple. Obviously, the number of tuples and distinct locations coincide in the single fault case. In all cases, unless redundancy caused by fault masking occurred as described next, the algorithm always returned a tuple with the actual set of fault locations.

Fault masking denotes the situation where some stuck-at fault(s) is not observable due to the presence of others. In a simple-minded example, a stuck-at fault on output of a gate renders all other stuck-at faults that may occur on the inputs of the same gate. More complicated situations are introduced by circuit topology and internal logic redundancy(ies). Fault masking did not occur in our experiments on the ISCAS'85 combinational circuits. This was not the case with the ISCAS'89 sequential circuits where fault masking occurred in more than 30% of the cases when 4 faults were injected, on the average. In these cases, the algorithm returned with triples or pairs that fully explain the fault behavior. The undetected fault(s) is considered redundant in the presence of other faults, and not reported in the table.

A candidate site is obtained by grouping together the stuck-at faults that are locally equivalent or in the same vicinity. An example of locally equivalent faults could be a stuck-at-0 on one input of a AND gate and a stuck-at-0 on its output. Two stuck-at faults lines that are connected by either a buffer or an inverter are considered in the same vicinity. They are so defined because there is high likelihood that these candidates are located closely in the chip and a test engineer can examine them all with one drilling. The average numbers of candidate fault lines returned by the algorithm are plotted against the number of faults initially injected in Fig. 2. This figure as well as Table 1 confirm that the pro-

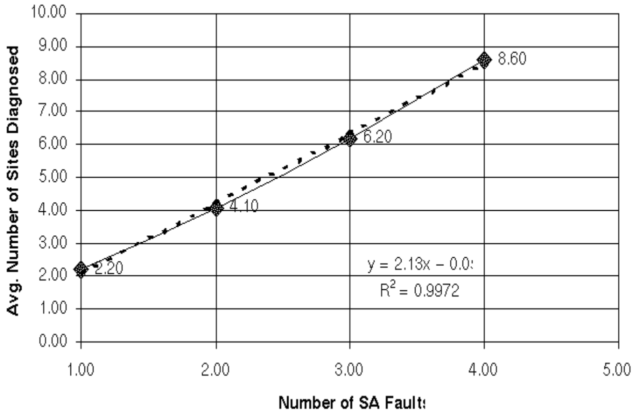

Figure 2. The average number of diagnosed candidate sites

posed algorithm exhibits excellent resolution and justify the effectiveness of an iterative diagnosis
and fault modeling approach.

We also observed, as Theorem 1 suggests, that the method is heavily biased towards faults with high excitation measures. In other words, the method is more likely to perform well in the situations where the design is corrupted with faults that are easy to excite (even if they are not observable to the primary outputs) and their V_i sets (Theorem 1) overlap. It is among our future plans to further experiment and measure the parameters involved with these heuristics so that we can improve on the overall performance.

The incremental approach has the advantages of being applicable as a diagnostic framework to other faults and errors as long as a suitable fault model is devised. In the case of stuck-at faults, this task is greatly simplified because the fault model is small and compact. We are working on extending this work to other types of physical faults.

5 Conclusions

We presented a simulation-based method for diagnosing multiple stuck-at faults. The method identifies multiple faults through a sequence of interleaving diagnosis and fault modeling steps, each of which strives to suggest one fault/line combination that accounts for a certain percentage of the erroneous outputs. Our experiments demonstrated the effectiveness and accuracy of the algorithm, which scales very well with increasing number of faults. This algorithm can be of valuable use to the industry in debugging faulty logic chips. In the future we plan to investigate its application to diagnosis of other physical faults.

References

[1] M. L. Bushnell and V. D. Agrawal, *"Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits,"* Kluwer Academic Publishers, 2000.

[2] I. Hamzaoglu and J. H. Patel "New Techniques for Deterministic Test Pattern Generation," in *Proc. IEEE VLSI Test Symposium, pp. 446-452,* 1998.

[3] S. Y.Huang, "Towards the Logic Defect Diagnosis for Partial-Scan Designs," in *Proc. of IEEE ASP-DAC, pp. 313-318,* 2001.

[4] D. Nayak and D. M. H. Walker, "Simulation-Based Error Diagnosis and Correction in Combinational Digital Circuits," in *Proc. IEEE VLSI Test Symposium*, pp. 70-78, 1999.

[5] I. Pomeranz and S. M. Reddy, "On correction of multiple design errors," in *IEEE Trans. on CAD*, vol. 14, pp. 255–264, February 1995.

[6] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," in *Proc. of Int'l Conference on Computer Design, pp. 328-333,* 1992.

[7] H. Takahashi, K. O. Boateng and Y. Takamatsu, "A New Method for Diagnosing Multiple Stuck-at Faults using Multiple and Single Fault Simulations," in *Proc. IEEE VLSI Test Symposium, pp. 64–69,* 1999.

[8] A. Veneris, and I. N. Hajj, "Design Error Diagnosis and Correction Via Test Vector Simulation," in *IEEE Trans. on Computer–Aided Design,vol. 18, no. 12, pp. 1803–1816,* December 1999.

[9] A. Veneris, B. Liu, M. Amiri and M. S. Abadir, "Incremental Diagnosis and Logic Debugging of Multiple Faults and Errors," in *Proc. of DATE,* 2002 (to appear).

[10] S. Venkataraman and W. K. Fuchs, "A deductive technique for diagnosis of bridging faults," in *Proc. IEEE ICCAD*, pp. 562–567, 1997.

[11] S. Venkataraman and S. B. Drummonds "Poirot: Applications of a Logic Fault Diagnosis Tool," in *IEEE Design and Test of Computers, pp. 19-30,* Jan.-Febr. 2001.