

Incremental Fault Diagnosis

Jiang Brandon Liu, *Member, IEEE*, and Andreas Veneris, *Member, IEEE*

Abstract—Fault diagnosis is important in improving the circuit-design process and the manufacturing yield. Diagnosis of today's complex defects is a challenging problem due to the explosion of the underlying solution space with the increasing number of fault locations and fault models. To tackle this complexity, an incremental diagnosis method is proposed. This method captures faulty lines one at a time using the novel linear-time single-fault diagnosis algorithms. To capture complex fault effects, a model-free incremental diagnosis algorithm is outlined, which alleviates the need for an explicit fault model. To demonstrate the applicability of the proposed method, experiments on multiple stuck-at faults, open-interconnects and bridging faults are performed. Extensive results on combinational and full-scan sequential benchmark circuits confirm its resolution and performance.

Index Terms—Circuit simulation, fault diagnosis, open-interconnect, very large scale integration (VLSI).

I. INTRODUCTION

TODAY'S world has been revolutionized by the rapid advancement of very large scale integration (VLSI) integrated circuit (IC) engineering. As the IC process technology becomes more complex and minimum feature size approaches nanometer range, manufacturing quality and yield are becoming more sensitive to physical defects. These defects are usually caused by mask contamination, process variation in fabrication, and spurious material [7], [10]. Defects can be modeled on the logic level by a fault that affects single or multiple circuit lines, and produces failing output responses for one or more input test vectors. Unraveling the location and cause of the defect, a process known as *failure analysis*, helps improve the circuit design and manufacturing process leading to a lower cost, improved yield and shorter time-to-market.

Failure analysis usually consists of two tasks: *fault diagnosis* and *defect analysis*. Fault diagnosis uses the observed failing responses and the structure of the circuit under diagnosis (CUD) to search for locations that are potentially faulty. This information is used in defect analysis, where the CUD is physically examined to determine and/or eliminate the mechanism of the failure [7], [10]. Because physical examination is inevitably slow and resource intensive, the efficiency of failure analysis depends on the *resolution* (i.e., accuracy) of diagnosis.

Single-fault diagnosis is a well-studied problem with various linear-time techniques [10]. Although helpful, single-fault diagnosis may not be adequate for defects in modern devices that tend to cluster and affect multiple lines in the failing chip [2], [3], [8], [12], [13], [20], [21]. This is also confirmed in recent experiments from a real-life diagnosis environment, which show that more than 41% of defects found in failing chips cannot be diagnosed using the single stuck-at fault model [8]. Diagnosis of such circuits is a difficult problem because, in theory, the solution space grows exponentially with the number of defective circuit lines [7], [20], [24]. Furthermore, a robust diagnosis algorithm should be able to identify defects whose behavior may not be modeled accurately by a fault model [2]–[4]. Clearly, a *brute-force approach* that exhaustively enumerates all possible locations will need to search a prohibitively large solution space.

To meet these challenges, a simulation-based effect-cause *incremental diagnosis* approach for clustered defects that affect multiple circuit lines is proposed [17], [25]. Incremental diagnosis is an iterative process where a single faulty location is identified at each iteration. Fault effects are forced on this line and its fanout cone. These effects can capture either that of a specific fault model (*fault-modeled diagnosis*) or that of any fault model (*model-free diagnosis*). Model-free diagnosis is performed by simulating a logic unknown [4] on the candidate fault line. This type of diagnosis is important when the defects' behavior cannot be accurately described by existing fault models [2], [4]. Incremental diagnosis is repeated until the netlist with forced values emulates the behavior of the faulty chip. Compared to previous incremental approaches [3], [8], [12], [13], the proposed work differs in the novelty of its algorithms, heuristics, and decision process when it searches the solution space.

To demonstrate the applicability of the proposed work, prototype computer-aided design (CAD) tools are developed for multiple *stuck-at* fault, *open-interconnect* fault, and *bridging* fault diagnosis. We select stuck-at faults because in multiplicity, they can model other types of faults [19] and various types of design errors [24]. Therefore, efficient model-based diagnosis algorithms for multiple stuck-at faults may help find solutions for other fault/error models as well [10].

Open-interconnect is caused by a break in interconnect wiring due to material or processing defects, electromigration, and thermal stress [5], [11]. It has been reported that it is the dominant failure in field returns of process chips [22]. Furthermore, the total length of interconnect wires grows rapidly as the International Technology Roadmap for Semiconductors [9] in Fig. 1, predicts. This trend may increase the failure rates attributed to opens in the near future. Open-interconnect is a fault with a complex behavior [5], [11], [19], [26]. To tackle this complexity model-free incremental diagnosis is used.

Manuscript received April 28, 2003; revised September 20, 2003 and March 30, 2004. This research was supported by a grant from Communications and Information Technology Ontario (CITO). This paper was recommended by Associate Editor N. K. Jha.

J. B. Liu is with the High-Performance Tools and Methodology Group, Freescale, Austin, TX 78729 USA (e-mail: brandon.liu@freescale.com).

A. Veneris is with the Department of Electrical and Computer Engineering and the Department of Computer Science, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: veneris@eecg.toronto.edu).

Digital Object Identifier 10.1109/TCAD.2004.841070

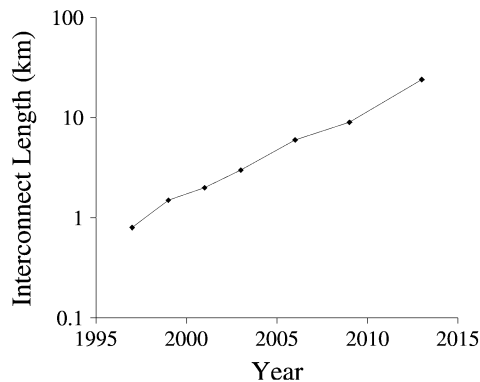


Fig. 1. ITRS on interconnect length.

An extensive suite of experiments using the incremental diagnosis method presented here confirm its efficiency in terms of runtime and resolution. Therefore, it can act as a cost-effective front-end tool to failure analysis to help reduce the number of suspect lines the test engineer has to probe.

This work is organized as follows. The next section describes the motivation and goals of incremental diagnosis. It also outlines the basic components of the algorithm as a sketch for an operating incremental diagnosis framework. In Section III, this framework is tailored to the diagnosis of multiple stuck-at faults (model-based) and multiple open-interconnect (model-free) faults. Experiments for a variety of faults are found in Section IV and the conclusion follows in Section V.

II. INCREMENTAL DIAGNOSIS

Incremental diagnosis is an iterative process that identifies one faulty location in each pass. Upon completion, it returns a set of N circuit lines, where *fault effects* can be forced to emulate the observed faulty CUD behavior for all input test vectors used. Fault effects are specified either by a set of predetermined fault model(s) or by using the logic unknown value(s). A logic unknown conservatively captures *any* faulty behavior on the line and it may diagnose defects whose behavior cannot be accurately fault modeled [2], [4].

Section II-A gives the motivation behind incremental diagnosis. An operational framework for the overall procedure is described in Sections II-B, –II-E. We use this framework to develop a model-based diagnosis tool for multiple stuck-at faults and a model-free one for multiple open-interconnect faults in Section III. Experiments demonstrate the applicability of the approach for these fault types and for cases where the circuit under test is corrupted with different defect types.

A. Motivation

As transistor density increases and feature size reduces, single-fault diagnosis may not be adequate for modern designs, where compound defects can affect more than one line in the circuit. Recent empirical data from a real-life design environment for 453 failing devices show that 41% of the defects found cannot be modeled with a single fault [8]. Additionally, 22% of the remaining 59% defect cases cannot be modeled using the single stuck-at fault model. Therefore, in more than 60% of the cases where a chip is returned for defect analysis, multiple

fault diagnosis is required and the classical single stuck-at fault model may be inadequate.

Efficient multiple-fault diagnosis algorithms are useful in other areas of the VLSI design cycle as well. In design-error diagnosis and correction [1], [25], [28], the designer uses diagnosis techniques to debug a netlist that fails verification due to the presence of a few bug(s). These bugs can have a low cardinality if they occur during logic synthesis or they may have large multiplicity as in the extraction of error debugging for test-model generation [28]. Logic debugging is a task that is strongly related to the more general problem of *engineering change* [16], where one is required to modify a netlist to meet a new specification. Common ways to carry out engineering changes use a traditional diagnosis/correction debugging approach [16]. In all cases, diagnosis for logic debugging and engineering change is a procedure symmetric to the one examined here [20]. Furthermore, diagnosis tools are utilized in *design rewiring* for design optimization [23]. The optimization gain of these methods depends on the ability of the underlying tool to perform multiple fault diagnosis [23]. Therefore, advances in multiple-fault diagnosis may aid logic debugging, engineering change and design rewiring.

Multiple-fault diagnosis is a challenging problem because, in theory, the search space grows exponentially with an increasing number of faulty lines [7], [24]. Additionally, straightforward implementations of a single stuck-at fault algorithm may not capture defects on multiple locations with effects that converge and/or mask each other [7], [3], [8], [17], [25].

To illustrate the last point, an experiment is performed to show multiple-fault interaction in combinational ISCAS'85 and ITC'99 benchmarks and full-scan versions of ISCAS'89 circuits. Open faults are randomly selected and injected one by one, and the number of erroneous primary outputs is recorded after each fault. To model open interconnects, we use the fault model presented in [26] which is also described in Section III-B of this paper.

In almost one third of the runs, the number of erroneous primary outputs does not increase monotonically as new faults are injected in the circuit. We present these cases in Table I. For each run, its fault injection sequence and the number of erroneous primary outputs is shown in the table. This phenomenon indicates the presence of fault convergence and fault masking that may misguide algorithms designed exclusively for single faults. A similar experiment from [18] using design errors also confirms the presence of error effect interaction with an increasing number of errors. Design errors are relevant to this work because they can be modeled using multiple stuck-at faults [1].

To meet these challenges, the incremental diagnosis method described in this paper uses linear-time algorithms to identify one faulty location one at a time and various heuristics and data structures that guide the search in the solution space. The rationale behind the approach comes from the fact that often, the majority of the failing input vectors can be attributed to a single faulty location [3], [8], [17], [25]. The following computation describes a desired runtime behavior for incremental diagnosis.

Consider a circuit with L lines and N distinct faulty lines, where $L \gg N$. Further, let K be the maximum number of fault manifestations on a single line. For example, in stuck-at

TABLE I
NONMONOTONIC BEHAVIOR OF FAILING PRIMARY OUTPUTS

circuit	first fault		second fault		third fault		fourth fault	
	location	EPO	location	EPO	location	EPO	location	EPO
c1908	G1323	2278	G1261	2272	G1022	2234	G1067	2218
	G1113	2293	G1296	2292	G1317	2291	G806	2295
	G924	6121	G1169	6118	G1256	5930	G1392	5925
C6288	G4596	909	G4355	903	G3482	910	G3612	910
	G3784	902	G4499	903	G4560	901	G2648	918
	G2831	7490	G2704	7501	G4773	7491	G3322	7512
S9234	G3050	64950	G3670	64951	18240	65212	G4703	64956
	G4563	64953	G5675	64741	G2927	64740	G6258	64506
	I7355	65199	I2162	65460	G3758	65201	I8863	65451
B17	U16434	237121	U15489	237076	U11905	237038	U12416	237220
	U13493	236936	U8766	236936	U17333	236622	U11631	236625
	U11524	118379	U9061	118380	U10842	118380	U12225	118456
B20	U5999	79707	U4705	79704	U5641	79701	U6369	79718
	U4274	79708	U5540	79716	U9266	79882	U4420	79881
	U6833	79711	U8358	79884	U8500	79883	U4406	79883
B21	U6929	79742	U8476	79657	U8620	79716	U4466	79715
	U3528	79698	U9324	79541	U6465	79709	U7189	79709
	U9027	79865	U7292	80015	U6921	80049	U4259	80048
B22	U12674	113792	U6147	113794	U9737	113794	U11715	113710
	U10435	113781	U9692	113790	U5203	113629	U11119	113629
	U8399	113624	U4988	113493	U8262	113502	U11718	113516

diagnosis, K is 2 because we can have stuck-at 0 or stuck-at 1 on the line. If the faults are identified incrementally and the computation at each iteration remains proportional to the circuit size, the total solution space probed equals $\alpha^N \times N \times L \times K$. If a cycle-based simulator is used that needs τ time units for a single vector simulation then the total time for diagnosis is $\alpha^N \times \tau \times |V| \times N \times L \times K$ time units.

In the above expressions, α is the average cost incurred by a wrong decision during each stage of the incremental diagnosis. Ideally, α is 1 and a fault is identified at each iteration. In such a case, the complexity reduces to a linear factor in terms of the number of faulty lines. In the worst theoretical case, α is polynomial to N and the complexity of the solution increases exponentially. Experiments show that α is upper-bounded by a small constant for the approach developed here.

B. Definitions and Algorithm Overview

Incremental diagnosis takes as inputs a CUD, its structural specification netlist, a set of test vectors V , including some with failing responses, and an integer N , which indicates the maximum cardinality of faulty locations to search. The netlists considered are combinational ones consisting of logic primitives AND, OR, NOT, NAND, and NOR, and full-scan sequential circuits with a fault-free scan-chain. It returns sets of circuit lines, each of cardinality N or less, that some fault effects on them can explain the faulty behavior for the failing device. Due to fault equivalences, the solution to diagnosis may not be unique. Since the number of faults is unknown prior to the diagnosis, N is specified by the user as a conservative guess. If the algorithm fails to return a solution (i.e., the number of faulty lines exceeds the initial guess), it automatically increases the value of N and restarts until a solution is found.

Definition 1: A *diagnostic configuration* is a partially diagnosed circuit during diagnosis. It is represented by the struc-

tural circuit netlist, injected faults, and the current logic simulation values stored in an indexed bit-list (see below). If at least one of its primary outputs has different simulation value(s) than the CUD, the configuration is said to be *active*. Otherwise, it is called *inactive* and the design behaves identically to the specification for all the input test vector stimuli.

For an input test vector v , a line whose value changes in the presence of a fault f is said to be *sensitized* to the fault f by vector v . A path consisting of sensitized lines is called a *sensitized path*. In a diagnostic configuration, if a primary output does not have the same logic value as the CUD, it is called an erroneous primary output (EPO). Otherwise, it is a correct primary output (CPO). Logic simulation values of all input test vectors are kept in two indexed bit-lists for every line in the circuit as in [24]. One bit-list corresponds to nonfailing vectors and the other to failing vectors. Both assist diagnosis and they are properly updated at each diagnostic configuration.

Definition 2: A circuit line under consideration by diagnosis is referred to as a *suspect* line. The combination of a suspect line and a fault model on this line is called a *candidate*. A set of candidates is called a *candidate tuple* (pair, triples, etc.) and the candidate tuple(s) returned by the algorithm is called a *solution tuple(s)*.

Fig. 2 illustrates the overall flow of the approach. The algorithm proceeds by iteratively identifying suspect locations one at a time. First, path-trace routine is performed to quickly prune the diagnosis space (Section II-C). Suspects are ranked according to theorems and heuristics found in Section II-D and the top one is selected. A fault effect is injected on the line (i.e., the logic values that reflect the fault effect are forced on the line) in the netlist and it is resimulated to yield a new diagnostic configuration. This completes *one iteration* of the algorithm. Subsequent iterations are conducted on the diagnostic configuration until it becomes inactive. All candidate lines selected in this process make up a solution tuple.

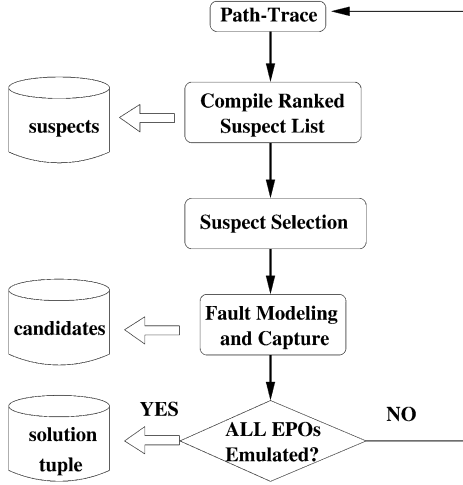


Fig. 2. Incremental diagnosis flow.

The flow above is ideal in the sense that it assumes after each iteration an original (or equivalent) faulty line is successfully found. In practice, a number of suspects are discovered. Due to fault convergence and fault masking, ranking does not guarantee that the original fault(s) is placed atop the list. To accommodate this, the algorithm bases its decision flow on a *search tree*, which is described in Section II-E. This tree allows the algorithm to search the solution space efficiently and capture the actual and/or equivalent fault locations.

C. Path-Trace

Path-trace is a linear-time routine in the effect-cause direction [27], which is similar to critical path tracing [10]. It starts from an EPO and pessimistically marks lines that may belong to a sensitized path. If the output of a gate has been marked and the gate has one or more fanin(s) with controlling values, then one of the controlling fanins is marked. If a gate has all fanins with noncontrolling inputs, then all fanins are marked. Finally, if a branch is marked, then the stem of the branch is marked. An example of path-trace is illustrated in Fig. 3, where it starts from EPO O1 and it marks lines with an asterisk “*.”

Path-trace is important in diagnosis due to the following theorem [24] that stipulates that each run of path-trace marks at least one line from each solution tuple.

Theorem 1: Let G be a CUD with N faulty locations and $L = l_1, l_2, \dots, l_N$ be any solution tuple. The set of lines marked by path-trace for some failing vector contains at least one line from L .

Path-trace is conducted over multiple failing vectors and the number of times it visits each line is recorded. All lines marked at least once are in the list of current suspects. Path-trace is generalized in Section III-B2 to accommodate model-free diagnosis.

D. Suspect Ranking

Each iteration of incremental diagnosis needs to discover only one candidate line. Even though the suspect list returned by path-trace is much smaller than the number of circuit lines, it

is still inefficient to examine each of its members exhaustively. To shorten this list, the following corollary stipulates a lower bound for how frequently a line needs to be marked so it qualifies. The corollary is an immediate consequence of Theorem 2 that follows.

Corollary 1: In a circuit with N defective lines, if path-trace is conducted for V^{err} failing input vectors, then one or more line(s) from each solution tuple will be marked at least $(|V^{\text{err}}|)/(N)$ times.

The number of suspect lines that pass this simple screening test is usually small. These lines are visited in descending order of path-trace counts and a faulty effect is injected for simulation. If a fault model is used, depending on its cardinality, a wrong selection may still impede the performance with the costly search of nonsolution space. Therefore, in model-based diagnosis Theorem 2 gives a necessary condition a fault model must obey.

Theorem 2: Let circuit with N defective lines and let V^{err} be a set of failing input vectors for this circuit. Any valid fault model for suspect line l needs to complement at least $(|V^{\text{err}}|)/(N)$ entries in the bit-list for failing vectors of l .

Proof: Let l_1, l_2, \dots, l_N be a minimal set of lines such that some fault effects on these lines can explain completely the faulty chip behavior for the vectors in V^{err} . Define V_i^{err} to be the maximum subset of vectors from V^{err} that produce a faulty logic value on l_i and propagates this difference to some primary output(s) via one or more sensitized paths. By definition, each vector in V^{err} sensitizes one or more such paths. In other words, each vector in V^{err} is attributed to *at least* one subset V_i^{err} for some $i, 1 \leq i \leq N$. Using the pigeonhole principle, the minimum size of the set V_i^{err} with the maximum cardinality among all such sets cannot be less than $(|V^{\text{err}}|)/(N)$. In other words, there is at least one line l in l_1, l_2, \dots, l_N that has $(|V^{\text{err}}|)/(N)$ or more incorrect logic values in its bit-list. Therefore, any valid fault model for l needs to complement $(|V^{\text{err}}|)/(N)$ or more logic values in l 's faulty vector bit-list. \diamond

Despite its simplicity, experiments show that the theorem provides a reliable guide to fault-model selection in model-based diagnosis. This theorem is not used in model-free diagnosis, where faulty effects are captured via logic unknowns.

After an appropriate fault effect is selected for a line, its fanout cone is simulated to obtain the updated diagnostic configuration. In this new configuration, the sensitized paths of the diagnosed fault are captured (i.e., emulated) and reflected at the primary outputs. However, as experimentally confirmed in Section II-A, the number of EPOs may not monotonically reduce as we identify new faulty lines due to fault convergence and/or fault masking. To alleviate this problem, some leniency is allowed by the algorithm and a qualifying candidate may sensitize a small number of new paths to previously correct primary outputs. The rationale behind this heuristic is illustrated with an example.

Example 1: Fig. 3 depicts the situation where two faults on lines G_{12} and G_{9-15} sensitize two paths that converge in gate G_{17} with faulty values 0 and 1, respectively. All simulated pairs of values in this paper are fault-free/faulty circuit values. G_{12} masks G_{9-15} such that the primary output O2 remains correct. If the algorithm corrects the faulty value on G_{12-17} , it allows O2 to be sensitized to G_{9-15} and it introduces an additional

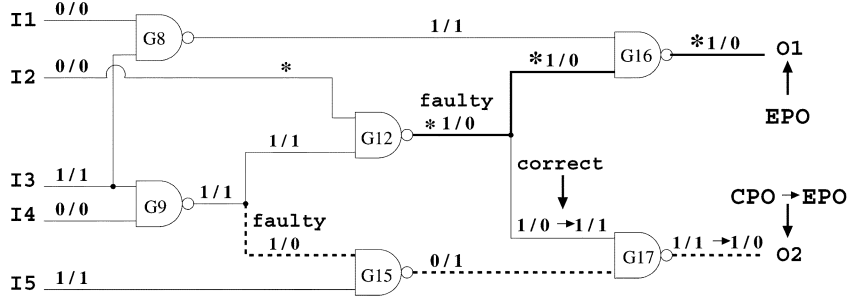


Fig. 3. Diagnosis in the presence of fault masking.

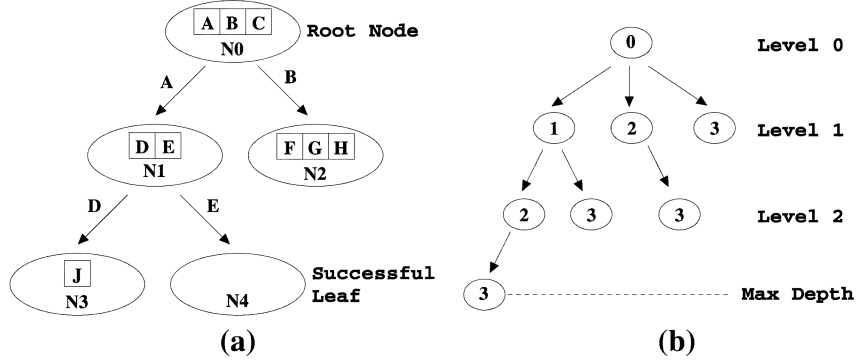


Fig. 4. Search tree and its traversal.

EPO. Notice, now that the fault on G_{9-15} is observed, it can be diagnosed in subsequent iterations of the algorithm.

The number of increased EPOs depends on the defect type/location and the circuit structure. An empirical limit of 5%–15% additional EPOs is used in implementation unless the netlist contains many XOR gates. For these circuits, experiments show that we may need to allow for a larger number (15%–20%) of new erroneous primary outputs to ensure success.

E. Search Tree

Because numerous suspect lines may exist for each active diagnostic configuration, a tree is maintained to facilitate the search of the solution space. In the example of Fig. 4(a), each node (represented by an ellipse) of the tree represents a diagnostic configuration. For each active diagnostic configuration (active node), a ranked list of suspect lines is compiled. Suspect lines are named by a single letter in this figure. The root node of the tree N_0 has three candidates: A , B , and C . Node N_4 is inactive and has no suspect lines.

To go from a node to one of its branches, a suspect line is chosen, represented by a labeled arrow. A suitable fault effect is selected for this line and its fanout cone is simulated to produce the new diagnostic configuration. Whenever an inactive node is reached, a solution is found (e.g., N_4). A depth bound N is set for the tree so it is not allowed to grow beyond it. An active node is automatically labeled as a “leaf” when the depth bound is reached. In effect, N indicates that there are at most N faults in the CUD. The suspect lines identified along the path between the root and an inactive leaf form a solution set, such as $\{A, E\}$ in Fig. 4(a).

Considering the large amount of suspect lines that can potentially model the fault effects at each iteration, a traversal order

that is a tradeoff between DFS and BFS is devised. The tree is traversed in breadth and depth simultaneously in *rounds*. During each round, all active nodes are extended by one child. Fig. 4(b) shows a search tree after four traversal rounds. Numbers in tree nodes indicate the round in which it is instantiated. This type of traversal is similar to the one described in [15] and it is guaranteed to reach a solution, provided sufficient time to enumerate the actual and all equivalent faulty locations.

A tree without a depth bound grows exponentially because the number of nodes doubles in each round. When a depth bound is asserted, the number of nodes $L^l(r)$ at tree level l after round r is described by

$$L^0(r) = 1 \quad (1)$$

$$L^l(0) = 0, \quad \text{for } l \neq 0 \quad (2)$$

$$L^l(r) = L^{l-1}(r-1) + L^l(r-1) \quad (3)$$

$$= \sum_{j=1}^l L^j(r-1) + 1. \quad (4)$$

Let $T^N(r)$ be the maximum number of nodes visited at round r in a search tree with depth bound N . $T^N(r)$ can be computed if we sum the nodes at each level of the tree

$$T^N(r) = \sum_{k=1}^N L^k(r) + 1 \quad (5)$$

$$= \sum_{k=1}^N \sum_{j=1}^k L^j(r-1) + \frac{N(N+1)}{2} + 1. \quad (6)$$

Fig. 5 illustrates the growth of the search tree for different values of N . It grows exponentially at the very beginning, but curbs linearly soon after. This theoretical upper bound is not

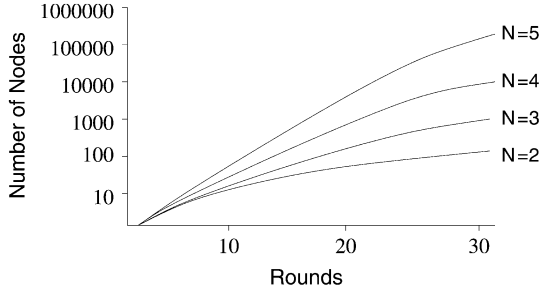


Fig. 5. Growth of search tree.

test vector	case I			case II		
	value	SA-0	SA-1	value	SA-0	SA-1
1	0	0	<u>1</u>	0	0	<u>1</u>
2	1	<u>0</u>	1	0	0	<u>1</u>
3	0	0	<u>1</u>	0	0	<u>1</u>
4	0	0	<u>1</u>	1	0	1
5	0	0	<u>1</u>	1	<u>0</u>	1
6	0	0	<u>1</u>	0	0	<u>1</u>

Fig. 6. Fault modeling in stuck-at diagnosis.

reached in practice and branches terminate as soon as solutions are found.

III. FAULT-DIAGNOSIS ALGORITHMS

The previous section outlines an incremental framework for multiple fault diagnosis. In this section, this general formulation is tailored to *model-based* and *model-free* diagnosis. We describe the model-based approach using stuck-at faults because they have a deterministic and simple behavior. The model-free algorithm is presented for open-interconnect faults because they have a rather complex behavior which is hard to model. Experiments are presented for these fault types as well as for the situation where both opens and bridges are simultaneously present in the circuit.

A. Stuck-At Fault Diagnosis

Since manifestation of a stuck-at fault consists of only two possible cases, the model-based algorithm does not follow the tree traversal from Section II-E, but it visits the tree with depth bound N in a depth-first manner. To improve performance, it considers lines with candidate stuck-at faults that are structurally fault collapsed [10].

As in Fig. 2, each iteration starts with a path-trace that marks suspect lines. For each one of them, it decides on the stuck-at fault model(s) to inject and simulate using Theorem 2. When the behavior of the CUD is completely emulated for the set of test vector stimuli, a solution tuple is found.

Example 2: In Fig. 6, the logic values on two different suspect lines are compared against the two stuck-at values. Suppose it is suspected that there are three faults in the circuit. By Theorem 2, a valid fault model inverts local logic values for at least $6/3 = 2$ vectors. In the table, inverted values for each fault case are underlined. In case I, stuck-at 0 fault inverts net value for one vector and stuck-at 1 fault for five vectors. Stuck-at 1 model is chosen for this suspect line and stuck-at 0 does not qualify. For case II, it can be shown, both faults qualify.

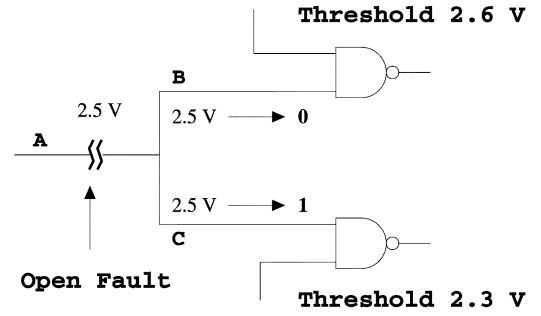


Fig. 7. Open fault on a stem.

B. Open-Interconnect Fault Diagnosis

Open-interconnect is a fault of great interest today. Its behavior depends on many physical circuit parameters—a fact that may prevent an accurate deterministic logic fault model. A physical open defect on an interconnect can cause different logic-level behaviors. 1) It may cause some lines to float. 2) It may slow down signal propagation. 3) It may amplify the effect of crosstalk [19]. The focus here is on opens that cause floating behavior.

A major difficulty in diagnosing open-interconnects is the absence of a simple deterministic logic fault model. As illustrated in Fig. 7, a stem with an open can have its value interpreted differently by its branches even for the same vector [5], [11]. This behavior can be seen as having all fanout branches of a stem behave at random and independent from each other. For a stem with t fanouts, 2^t different combinations of values may occur [17], [19], [26]. Although this net diagnostic model, originally proposed by [26], covers all possible faulty behavior, its size is often large for traditional diagnosis methods. To handle such complexity, we tailor the incremental approach to a model-free environment in which fault effects are captured with a logic unknown(s).

1) Algorithm Overview: Unlike diagnosis algorithms that explicitly model fault effects [3], [13], [21], [25], model-free diagnosis [2], [4], [17] simulates a *logic unknown* X on candidate lines to capture *any* faulty behavior as well as interaction between different fault effects. The proposed algorithm differs from model-free approaches such as [4] as it works incrementally. It does not also assume a region-based fault locality model. Instead, faults can be (structurally) located anywhere in the circuit. The following example illustrates the basic idea behind model-free incremental diagnosis.

Example 3: Fig. 8(a) displays a circuit with two open faults located on lines G_8 and G_9 . Both primary outputs are erroneous for this input vector. Suppose incremental diagnosis locates fault on G_9 first. The diagnostic configuration after placing an X on G_9 and simulating its fanout cone is shown in (b). It is seen that X propagates to G_{17} , which is originally an EPO. In the second iteration of diagnosis, all the lines with X in Fig. 8(b) are ignored and G_8 is identified as a potential fault location. In the configuration obtained after simulation of X on line G_8 [Fig. 8(c)], every EPO has an X on it. Consequently, $\{G_8, G_9\}$ is identified as a candidate tuple.

Diagnosis proceeds in two phases. At first, model-free incremental diagnosis (MFID) identifies a set of candidate tuples. If

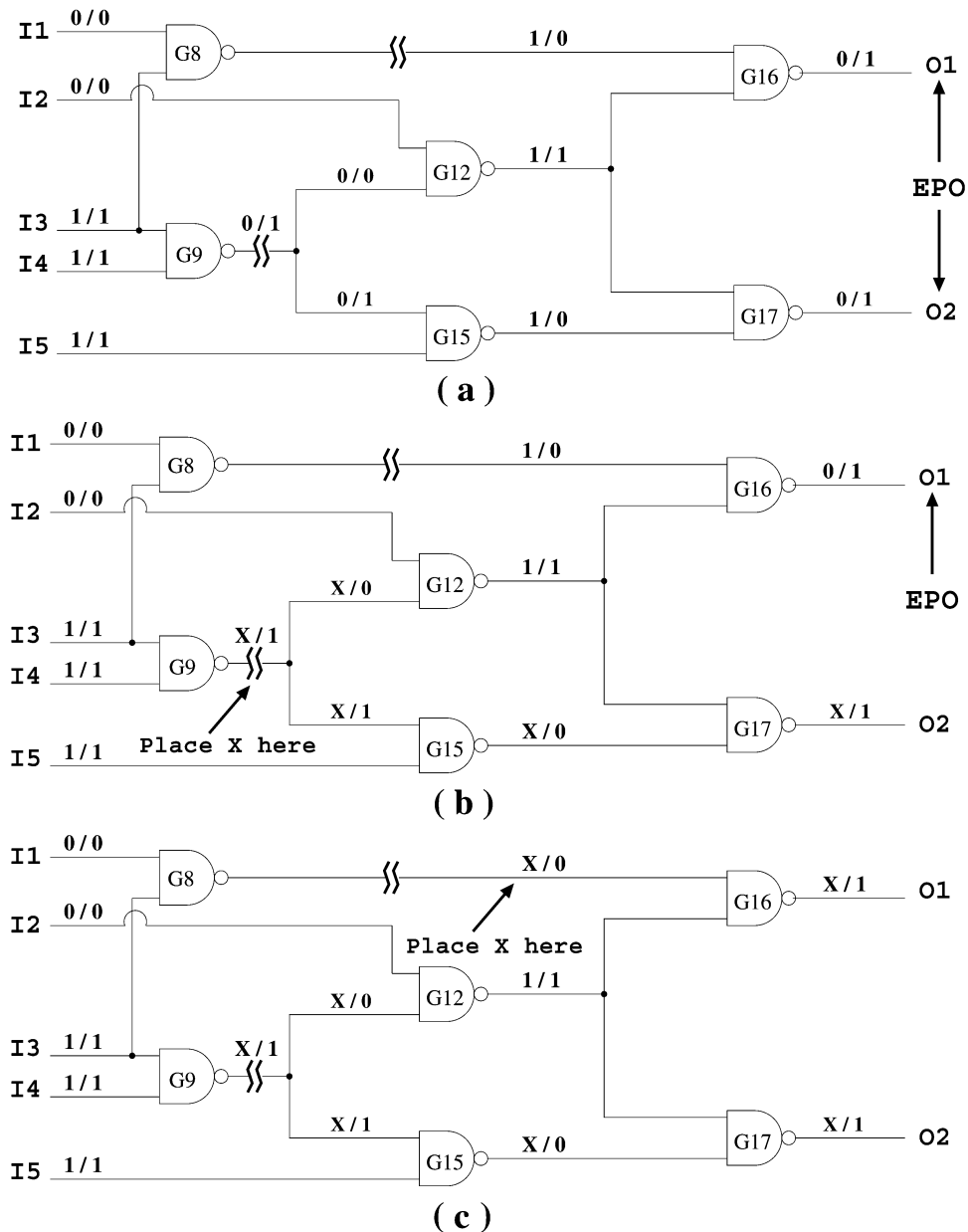


Fig. 8. Model-free diagnosis for two open faults.

an X is forced on all member lines of each tuple simultaneously, it propagates to all the EPOs. Its flow is similar to that in Fig. 2 with few modifications. Path-trace is extended to a 3-valued path-trace routine described in the next section. A novel suspect list compaction scheme collapses lines with similar (logic unknown) behavior into a single class to speed up the method (Section III-B3). The top ranking class is selected and X is forced on a representative member of the class to complete one iteration of MFID. Multiple iterations are conducted until X propagates to all the EPOs. The conservative nature of logic unknown may lead to large solution sets. The second phase, presented in Section III-B4, prunes this list of candidates with generalized fault simulation (GFS).

2) *3-Valued Path-Trace*: Routine 3-valued path-trace marks lines using ternary logic simulation. The path-trace routine described earlier is augmented with the following rule.

Rule: Never mark a line with X on it.

Path-trace may deduce information by following a trail of possibly erroneous values in the circuit. Due to the conservativeness of the unknown value, lines with unknown values can *never* increase the number of EPOs. Therefore, a line with an X can provide no information to path-trace and Theorem 1 holds for this routine as well.

Example 4: The operation of 3-valued path-trace is demonstrated in Fig. 9. Two faults converge on a NAND gate. In the initial diagnostic configuration, the trace marks the first fault [Fig. 9(a)] but the output of the gate remains unchanged, due to the remaining faulty controlling value as shown in Fig. 9(b). Invoking 3-valued path-trace again marks the second fault and X propagates past the gate, as shown in the last two figures. Hence, the gate will not be marked again.

3) *Suspect Compaction*: Like path-trace, the set of candidate lines marked by 3-valued path-trace needs to be pruned due to the pessimistic nature of the procedure. A suspect class-com-

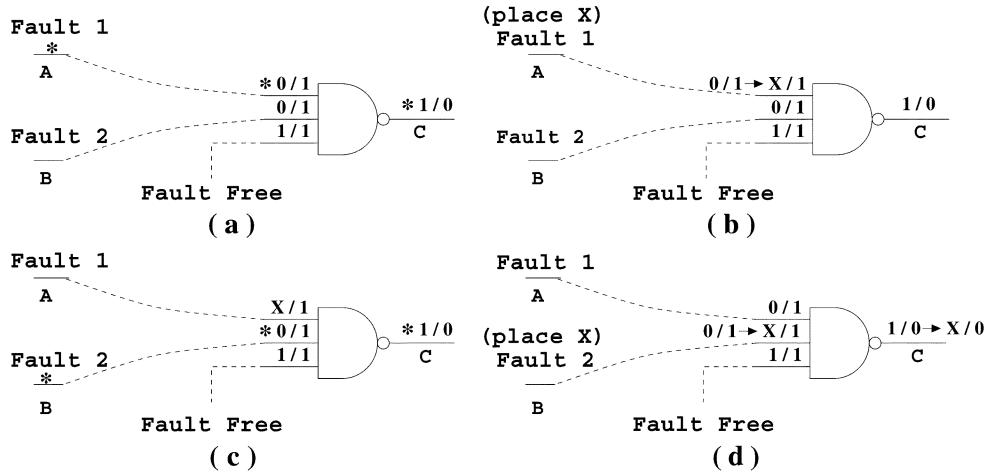


Fig. 9. Three-valued path-trace example.

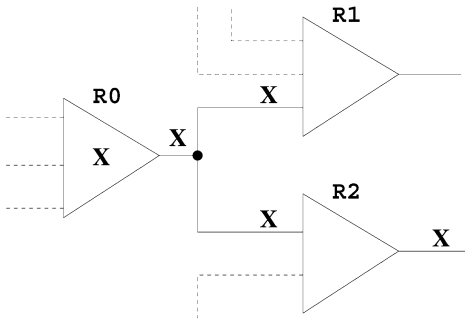


Fig. 10. Suspect compaction example.

paction method is devised to reduce the number of these candidates. Suspect compaction collapses lines with “similar” fault effects when an X is simulated at their fanout cone into the same class. It also ranks these classes with their potential to propagate X to EPOs. Suspect compaction reduces the size of the decision tree and fewer rounds are needed to capture faulty locations. The example that follows illustrates the idea.

Example 5: Let a triangle in Fig. 10 represent some fanout-free circuitry. Assume a logic unknown is forced on one of the lines inside $R0$ that propagates to its headline (i.e., the first stem that dominates $R0$ circuitry [10]). Also assume that this X propagates to the headline of $R2$ but not to the headline of $R1$. It is observed, an X on $R0$, or on $R2$, or on both lines have the same effect at the primary outputs. Additionally, the effect of simulating X at $R0$ “dominates” simulating X at $R2$ because it introduces more logic unknown values in the circuit. The algorithm *compacts* all lines of $R0$ ’s and $R2$ together by placing an X in the original faulty $R0$ line and simulating the effect in its fanout cone.

Pseudocode for the compaction algorithm is found in Fig. 11. Lines 9–12 of the algorithm give the compaction criterion: line M is in L ’s class if logic unknowns can always propagate from L to M for all test vectors turning into X ’s for any line inside the class. Clearly, M is in the fanout cone of L . The representative L of the class has the useful property that when simulating the unknown value on its fanout cone, it propagates the most number of X s to EPOs.

Compact(candidate list)

- (1) Sort candidate list ascendingly by the level of lines.
- (2) From the beginning of the list, repeat for each line L
- (3) if L does not belong to any compacted class
- (4) place X on L
- (5) for all the vectors, simulate the fanout cone of L
- (6) create a new compacted class C
- (7) insert L into C
- (8) examine each line M in the candidate list after L
- (9) if M has X value for all test vectors
- (10) place M into C
- (11) set L to be the representative member of C
- (12) compile output matching statistics and attach to C
- (13) restore the logic values in the circuit
- (14) sort all the classes according to their matching score

Fig. 11. Class-compaction algorithm.

4) *Generalized Fault Simulation (GFS):* The list of candidate class tuples returned by the model-free incremental diagnosis are converted to candidate tuples by computing the Cartesian product of all members of a class. The resulting list may be large and is pruned with GFS. This procedure generalizes the *error simulation* algorithm from [24].

For each candidate tuple, fault simulation emulates the effect of an arbitrarily complex fault on a line tuple. It accomplishes this by enumerating all possible combinations of logic values 0 and 1 on the lines of the members of the tuple for each vector. For n candidate lines, where stems of the tuple are replaced by their branches, fault simulation requires $2^n - 1$ simulations for each vector. A candidate tuple qualifies as location for an open fault(s) if for every failing test vector, it reproduces the observed behavior with at least one value combination.

Example 6: Recall in Example 3, tuple $\{G_8, G_9\}$ is returned by diagnosis expanded to $T = \{G_8, G_{9-12}, G_{9-15}\}$. This al-

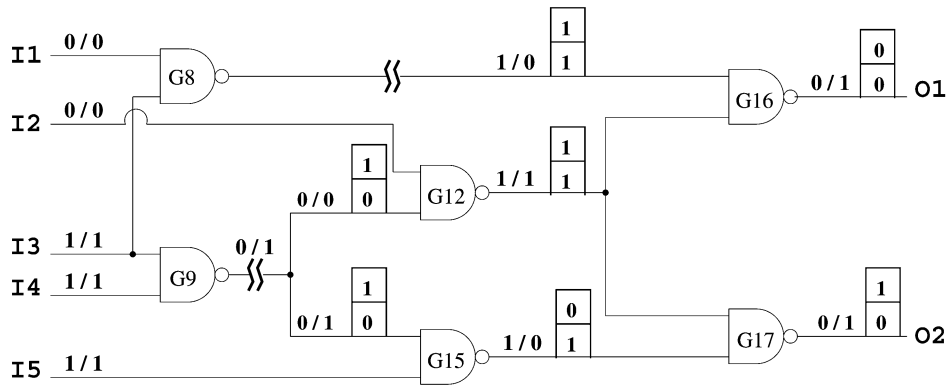


Fig. 12. Fault-simulation example.

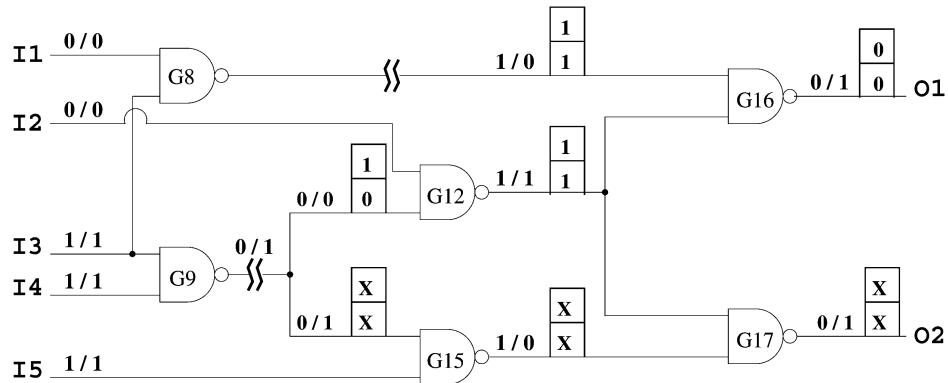


Fig. 13. Generalized fault-simulation example.

lows branches to float independently. There are 2^3 possible simulations, two of which are shown in Fig. 12. The values in the upper square boxes are simulation values when we inject values $\{1, 1, 1\}$ on the lines of T and the lower boxes when we inject values $\{1, 0, 0\}$. These values are potential fault manifestations on these lines. Both combinations produce the faulty behavior on O_1 , but only the lower one does for O_2 . As long as some manifestation produces the observed EPO behavior for each vector, GFS qualifies the tuple. In this case, $\{G_8, G_9\}$ qualifies GFS.

When the number of branches is large, it may be computationally expensive to simulate all applicable combinations of logic values for all candidate tuples [17]. To reduce this complexity, GFS places a logic unknown value on a subset s of the n lines and it enumerates only 2^{n-s} fault manifestations. It qualifies the tuple if for each vector some fault manifestation corrects all EPOs or it covers them with an X . The value of s depends on the number of branches for the stem under consideration. In implementation, we limit the number of simulations per fault site to 32 simulations (i.e., $n - s = 5$).

We omit the code for GFS, which is a direct extension of the one found in [24]. It is of interest to note that a similar technique is developed independently by [19], and implemented in a test generator for open faults in circuits with large fanouts. Therefore, this computational tradeoff seems to be inevitable when open faults are present on stems with many branches.

Example 7: In the circuit of Fig. 13, an unknown is placed on G_{9-15} and GFS is performed on G_8 and G_{9-12} . Again, the results of simulating only two fault excitation scenarios are shown in that figure. To qualify tuple $\{G_8, G_9\}$, the output O_2 is ig-

TABLE II
RANKING OF ORIGINALLY INJECTED FAULTS

circuit	stuck-at Fault			open Fault		
	1st	2nd	3rd	1st	2nd	3rd
C1908	1.4	1.9	3.1	1.8	2.2	3.2
C3540	2.3	2.5	4.0	1.5	2.3	3.9
C6288	3.2	4.1	4.7	2.7	4.4	6.0

nored because it has a logic unknown on it. O_1 is the only output that needs to be considered since it produces the faulty response.

IV. EXPERIMENTS

Experiments are conducted using ISCAS'85 and ITC'99 combinational circuits as well as full-scan versions of ISCAS'89 sequential circuits. The netlists are first optimized for area to mimic a faulty chip in an industrial testing environment. The number of circuit lines (including primary inputs/outputs and fully-scanned memory elements) is found in the second column of Table III.

Faulty chip behavior is emulated by injecting fault models into a gate-level implementation of the circuit specification. We perform three different types of experiments: 1) single and multiple stuck-at faults; 2) single and multiple open interconnects; and 3) simultaneous bridge and open faults.

In each experiment, the location of the fault is selected in the circuit at random, except for open-interconnects. More than 75% of the open faults are introduced at randomly selected stems with a large fanout count, since this presents a challenge for model-based diagnosis algorithms [17], [26]. Test vectors

TABLE III
RESULTS FOR STUCK-AT FAULTS

circuit	lines	1 fault		2 faults			3 faults			4 faults		
		sites	time	tuples	sites	time	tuples	sites	time	tuples	sites	time
C499	1338	1.1	0.7	2.8	3.4	0.7	7.3	5.8	0.9	3.3	5.9	29.6
C880	997	2.7	0.2	6.9	4.9	0.2	10	6.1	0.6	9.4	7.3	4.2
C1355	1284	3.7	0.2	2.3	3.0	0.8	4.5	5.5	10.9	9.3	7.3	44.9
C1908	983	1.0	1.0	2.3	3.2	2.7	12.1	6.5	0.9	7.3	6.8	16.7
C2670	1681	2.2	0.7	6.2	5.6	0.4	4.5	5.0	2.3	31.3	9.8	13.8
C3540	2358	1.2	2.0	4.9	4.5	0.7	7.2	6.1	29.7	14.3	8.3	67.6
C5315	4000	1.7	1.6	2.9	3.3	3.0	6.4	5.1	15.9	4.1	6.1	276.0
C6288	6326	1.1	3.4	2.2	2.8	2.9	5.9	4.7	14.6	1.5	4.5	61.3
C7552	5566	2.7	1.6	5.2	3.9	8.2	13.4	6.2	6.3	9.3	8.3	175.0
S838	798	2.2	0.2	4.0	3.3	0.4	7.2	5.0	1.8	214.7	15.3	1.01
S953	1150	1.9	0.3	2.6	3.4	0.6	2.3	2.3	13.8	96.1	17.2	49.4
S1196	1292	2.4	0.4	2.7	3.0	0.8	6.8	6.5	3.5	5.5	7.2	16.9
S1494	1454	2.9	0.2	7.0	5.3	0.3	18.9	9.0	1.3	5.3	6.3	17.2
S9234	4926	4.1	0.7	31.3	8.4	0.4	116	12.5	2.9	17.3	10.2	33.9

used in diagnosis are ATOM vectors [6] with 100% single stuck-at fault coverage and 1000–2000 random vectors. Ten experiments for each circuit and each fault case are performed. Experiments are conducted on a Sun Ultra 10 machine with 256 MB of physical memory.

In this presentation, *sites* is used to indicate the distinct locations returned by the algorithm. As explained in the introduction, this measure is useful because it narrows the task of the test engineer who probes these sites to unveil the cause of failure. Another important measure is *hit percentage* (h.p.) that counts the percentage of the actual faults within the set of sites returned. Since the value of N is unknown, the algorithm starts with small values and it restarts if it fails to return with a solution. Run-times presented in this Section denote the *accumulated* time (including restarts) for the algorithm and they are reported in CPU seconds.

A. Determination of α

To determine a suitable number of tree rounds, an upper bound for α is first determined empirically with some circuits. Recall from Section II-A, α is the number of wrong decisions an incremental diagnosis algorithm is allowed to do at each iteration. The size of the tree and the computational effort of the method directly relate to the choice of this parameter. Intuitively, the larger the α , the more expensive the method becomes, but it also increases its potential to identify all actual faulty locations.

To determine α , three stuck-at faults and three open faults are injected independently and their rank according to the heuristics presented in Section II-D is studied. The highest ranking fault is then removed and the experiment is repeated for the other two faults etc. The rank of each fault is tabulated in Table II. Since all faults rank within the top five candidates, a reasonable estimate for the value of α is 5. So, in diagnosing 1–3 faults, the algorithm is run for 5, 10, or 15 rounds respectively. This allows the search tree to have up to 5, 25, and 125 nodes, as shown in Fig. 5.

B. Stuck-At Fault

The results for multiple stuck-at fault diagnosis are presented in Table III. The second column contains the number of lines

in each circuit, including primary inputs, outputs, and registers. Column “*tuple*” is the average number of solution tuples returned by the algorithm. Obviously, the number of tuples and distinct locations coincide in the single fault case.

In all cases, the algorithm returns the tuple with the original faults, unless the injected fault(s) are redundant or they mask each other. Fault masking did not occur in our experiments on the ISCAS’85 combinational circuits. This is not the case with the ISCAS’89 sequential circuits where fault masking occurs in more than 25% of the cases in the four fault case. In these cases, the algorithm returns with triples or pairs that fully explain the faulty circuit behavior.

On average, the number of sites is about twice the number of faults injected. In other words, of every two sites, one is an originally injected fault. This reflects the excellent resolution of the algorithm. Note that the additional sites are caused by the presence of equivalent fault tuples, which cannot be eliminated by logic-level diagnosis methods alone.

C. Open-Interconnect Fault

Results for circuits with one open fault and the two distinct phases (MFID and GFS) of model-free diagnosis are summarized in Table IV. The final resolution and performance of the method is found in columns 6 and 7 of that table, respectively. Results on multiple opens are presented in Table V.

It is seen, all injected faults are found (100% hit percentage) in almost all circuits within a short time. For example, in Table IV, all circuits have 100 hit percentage except C6288. The 16-bit multiplier C6288 contains many fanouts and reconvergences, which create an unfavorable environment for diagnosis. This is because the large number of fanouts propagate the logic unknown to a large portion of the circuit. Consequently, many tuples qualify at each round. It is observed experimentally that it takes twice as many rounds as other circuits to capture the injected faults in this circuit.

Table IV also demonstrates the effectiveness of GFS in reducing the number of fault tuples. On average, it eliminates about two thirds of the candidate tuples by model-free diagnosis. By comparing the CPU time, it may appear that GFS can achieve better resolution in much less time than incremental model-free diagnosis. In fact, GFS only appears to be faster here because it

TABLE IV
RESULTS FOR SINGLE OPEN FAULTS

circuit	MFID			GFS		
	tuples	sites	time	tuples	sites	time
C432	36.5	6.3	0.9	16.0	6.3	0.1
C499	83.4	12.8	2.3	24.6	6.0	0.3
C880	32.0	7.8	0.4	11.6	6.2	0.2
C1355	37.0	10.5	2.0	14.8	5.7	0.3
C1908	77.6	13.0	1.5	10.5	6.0	0.2
C2670	41.0	8.0	3.9	15.3	8.0	0.4
C3540	47.9	5.5	17.0	22.0	5.5	0.8
C5315	37.0	8.0	5.1	13.0	3.5	1.4
C6288	67.0	16.5	25.2	6.5	3.0	2.2
C7552	65.3	12.7	7.5	22.8	9.4	2.2
S838	30.0	4.6	0.1	21.5	4.6	0.1
S1196	107.2	11.8	0.5	52.6	9.3	0.4
S1494	42.0	7.1	1.1	16.0	7.1	0.4
S9234	26.8	5.0	3.1	16.0	5.0	1.8
S38417	21.8	12.4	57.1	11.4	11.0	9.7
S38584	14.3	6.3	36.3	5.0	5.0	17.2
B17	56	21.25	183.8	30.1	20.5	50.5
B20	1902	20.2	306.0	10.5	6.4	39.4
B21	2150	43.5	300.2	6.2	2.8	92.7
B22	3650	33	437.6	8.9	4	209.2

TABLE V
RESULTS FOR TWO AND THREE OPEN FAULTS

circuit	2 Faults			3 Faults		
	sites	time	h.p.	sites	time	h.p.
C432	15.6	7.9	100	17.4	42.1	100
C499	30.0	30.8	100	47.8	354.0	100
C880	10.4	7.9	100	15.0	191.8	97
C1355	29.6	29.8	100	35.8	522.0	100
C1908	19.6	18.0	95	22.6	276.3	100
C2670	17.4	11.8	95	18.4	632.0	100
C3540	14.8	94.8	90	21.6	490.0	90
C5315	20.8	139.2	100	30.2	885.0	100
C6288	7.0	201.6	80	20.5	1442.0	87
C7552	17.0	60.5	100	22.2	898.0	93
S838	12.0	2.8	95	19.0	32.6	90
S1196	13.4	20.2	100	30.0	220.8	97
S1494	16.2	35.9	100	30.7	346.4	100
S9234	11.6	143.3	100	21.0	675.0	100
S38417	18.3	413.8	100	11.0	1455.0	100
S38584	14.3	465.7	100	8.3	1133.0	100
B17	11.7	905.7	93	28	2021.3	90
B20	30.0	1459.0	100	52.4	2968.4	97
B21	11.7	1642.7	100	45	3563.0	83
B22	16.3	1228.1	100	25	3657.3	100

examines much fewer suspect tuples than the model-free incremental diagnosis (MFID).

The effectiveness of the suspect list compaction heuristic to improve the performance of model-free diagnosis is also investigated. Model-free diagnosis is performed without suspect compaction and the ranking of the faults is reported in Table VI. When compared to the respective rankings in the last three columns of Table II, we observe that it improves the ranking of the original faulty lines significantly. This reduces the size of the tree and improves runtime performance with no sacrifice to the final resolution.

The average number of sites is about eight times the number of faults injected. Clearly, stuck-at fault diagnosis outperforms

TABLE VI
SUSPECT COMPACTION HEURISTIC

circuit name	suspect line ranking			suspect class ranking		
	1st	2rd	3rd	1st	2rd	3rd
C1908	24.5	30.3	91.7	2.6	4.7	10.2
C3540	31.1	55.6	44.0	2.9	3.6	7.8
C6288	57.0	60.2	55.3	7.4	10.1	22.5

TABLE VII
RESULTS FOR OPENS AND BRIDGES

circuit	1 bridge end/open			2 bridge ends/open		
	sites	time	h.p.	sites	time	h. p.
c432	16.7	1.8	100	24.8	3.5	100
C499	38.1	162.3	100	84.8	402.1	100
C880	11.5	20.8	100	17.6	85.7	100
C1355	32.4	62.0	100	56.8	158.4	100
C1908	48.1	45.9	100	136.3	97.6	93
C2670	16.9	37.2	100	35.8	98.6	100
C5315	30.6	146.3	100	55.8	289.1	100
C6288	10.6	317.2	96	46.5	615.2	87
C7552	46.8	178.0	100	95.5	266.6	93
S880	32.7	23.1	100	16.2	64.2	100
S1196	13.8	7.2	100	14.2	15.6	100
S1494	15.6	5.4	100	17.5	11.3	100
S9234	12.2	89.4	100	23.3	315.7	93
S38417	10.8	283.9	100	15.3	789.1	100
S38584	5.7	320.4	100	9.8	753.4	100

opens in resolution. This is because of the nondeterministic behavior of the fault and the way model-free diagnosis operates. It does not impose any condition on the faulty behavior, unlike stuck-at fault diagnosis that requires the same logic value to represent a stuck-at fault for all failing input vectors. Therefore, for open interconnects, there exist many more fault equivalences resulting in an increased number of sites.

D. Open and Bridging Faults

To evaluate the potential of model-free diagnosis further, we perform a set of experiments where both an open and a bridging fault are present and diagnosed in the circuit. We consider wired-AND and wired-OR nonfeedback faults as described in [10]. The location and type of the bridging faults are selected at random. Open faults are always inserted at stems since they are harder to diagnose. Results for this experiment are outlined in Table VII.

The first column contains the circuit name. The next three columns contain information in the case where the open and one endpoint of the bridge are diagnosed. To achieve this, we set the algorithm to look for two fault sites ($N = 2$). The last three columns contain data for the case when both endpoints of the bridge and the location open location are diagnosed by the algorithm ($N = 3$).

It is seen that the resolution for the first case is much better than the second one. This is because, in most experiments, the faulty behavior at two endpoints of the bridge is excited unevenly. This results in many equivalent candidates when searching for both endpoints of the bridge including the one which is not excited often. However, in a realistic diagnostic scenario, finding one line can significantly reduce the suspects for the other end of the bridge. This is usually performed by

processing layout information as explained in [10] and [14]. Therefore, we may conclude, these results further confirm the potential of the approach in an environment where multiple faults of different types are present in the circuit.

V. CONCLUSION

Fault diagnosis is crucial to reduce the cost and time to develop and manufacture IC chips. This work addresses the need for efficient and scalable diagnoses in today's complex VLSI design environment. It proposes an incremental framework for diagnosis that scales well with multiple faults. In this approach, faults are diagnosed one at a time with algorithms that have linear, to the circuit size, performance. Within this framework, diagnosis of two types of faults is examined; stuck-at fault and open interconnect fault. To capture the complex behavior of multiple opens, the idea of model-free incremental diagnosis is introduced. Experiments on circuits for these types of faults but also for faults of different types that are simultaneously present in the circuit confirm the scalability, performance, and resolution of the approach.

ACKNOWLEDGMENT

The authors acknowledge important technical contributions by H. Takahashi, M. S. Abadir, and M. Amiri in earlier versions of this work. They also thankfully acknowledge the associate editor and the anonymous reviewers of this paper for comments that improved its presentation.

REFERENCES

- [1] M. S. Abadir, J. Ferguson, and T. E. Kirkland, "Logic verification via test generation," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 7, no. 1, pp. 138–148, Jan. 1988.
- [2] R. C. Aitken, "Modeling the unmodelable: Algorithmic fault diagnosis," *IEEE Des. Test Comput.*, vol. 14, no. 3, pp. 98–103, Jul./Sep. 1997.
- [3] T. Bartenstein, D. Heaberlin, L. Huisman, and D. Sliwinski, "Diagnosing combinational logic design using the single location at-a-time (slat) paradigm," in *Proc. IEEE Int. Test Conf.*, 2001, pp. 287–296.
- [4] V. Boppana and M. Fujita, "Modeling the unknown! Toward model-independent fault and error diagnosis," in *Proc. IEEE Int. Test Conf.*, 1998, pp. 1094–1101.
- [5] C. Di and J. A. G. Jess, "On accurate modeling and efficient simulation of CMOS opens," in *Proc. IEEE Int. Test Conf.*, 1993, pp. 875–882.
- [6] I. Hamzaoglu and J. H. Patel, "New techniques for deterministic test pattern generation," in *Proc. IEEE VLSI Test Symp.*, 1998, pp. 138–148.
- [7] S. Y. Huang, "Toward the logic defect diagnosis for partial-scan designs," in *Proc. IEEE Asian-South Pacific Design Automation Conf.*, 2001, pp. 313–318.
- [8] L. M. Huisman, "Diagnosing arbitrary defects in logic designs using single location at a time (slat)," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 23, no. 1, pp. 91–101, Jan. 2004.
- [9] International Technology Roadmap for Semiconductors (1997). [Online]. Available: <http://www.itrs.net>
- [10] N. Jha and S. Gupta, *Testing of Digital Systems*. Cambridge, U.K.: Cambridge Univ. Press, 2003.
- [11] H. Konuk, "Fault simulation of interconnect opens in digital CMOS circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1997, pp. 548–554.
- [12] D. B. Lavo, B. Chess, T. Larrabee, and I. Hartanto, "Probabilistic mixed-model fault diagnosis," in *Proc. IEEE Int. Test Conf.*, 1998, pp. 1084–1093.
- [13] D. B. Lavo, I. Hartanto, and T. Larrabee, "Multiplets, models, and the search for meaning: Improving per-test fault diagnosis," in *Proc. IEEE Int. Test Conf.*, 2002, pp. 250–259.
- [14] D. B. Lavo, T. Larrabee, and B. Chess, "Beyond the byzantine generals: Unexpected behavior and bridging fault diagnosis," in *Proc. IEEE Int. Test Conf.*, 1996, pp. 611–619.
- [15] R. H. Lewis and C. Papadimitriou, *Elements of Theory of Computation*. Englewood Cliffs, NJ: Prentice-Hall, 1991, sec. 1.6.
- [16] C.-C. Lin, K.-C. Chen, S.-C. Chang, and M. M. Sadowska, "Logic synthesis for engineering change," in *Proc. ACM/IEEE Design Automation Conf.*, 1995, pp. 647–652.
- [17] J. B. Liu, A. Veneris, and H. Takahashi, "Incremental diagnosis of multiple open-interconnects," in *Proc. IEEE Int. Test Conf.*, 2002, pp. 1085–1092.
- [18] I. Pomeranz and S. M. Reddy, "On correction of multiple design errors," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 14, no. 2, pp. 255–264, Feb. 1995.
- [19] S. M. Reddy, I. Pomeranz, H. Tang, S. Kajihara, and K. Kinoshita, "On testing of interconnect open defects in combinational logic circuits with stems of large fanout," in *Proc. IEEE Int. Test Conf.*, 2002, pp. 83–89.
- [20] A. Smith, A. Veneris, and A. Viglas, "Design diagnosis using boolean satisfiability," *Proc. IEEE Asia South Pacific Design Automation Conf.*, pp. 218–223, 2004.
- [21] H. Takahashi, K. O. Boateng, K. K. Saluja, and Y. Takamatsu, "On diagnosing multiple stuck-at faults using multiple and single fault simulations in combinational circuits," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 21, no. 4, pp. 362–368, Apr. 2002.
- [22] M. Tripp, "Open microphone-wanted: New test directions and practical bottle necks," presented at the Proc. IEEE Int. Test Conf., 2001.
- [23] A. Veneris and M. S. Abadir, "Design rewiring using ATPG," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 21, no. 12, pp. 1469–1479, Dec. 2002.
- [24] A. Veneris and I. N. Hajj, "Design error diagnosis and correction via test vector simulation," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 18, no. 12, pp. 1803–1816, Dec. 1999.
- [25] A. Veneris, J. B. Liu, M. Amiri, and M. S. Abadir, "Incremental diagnosis and debugging of multiple faults and errors," in *Proc. IEEE Design Test Eur.*, 2002, pp. 71–721.
- [26] S. Venkataraman and S. B. Drummonds, "A technique for logic fault diagnosis of interconnect open defects," in *Proc. IEEE VLSI Test Symp.*, 2000, pp. 313–318.
- [27] S. Venkataraman and W. K. Fuchs, "A deductive technique for diagnosis of bridging faults," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1997, pp. 562–567.
- [28] Y. S. Yang, J. B. Liu, P. Thadikaran, and A. Veneris, "Extraction error diagnosis and correction in high-performance designs," in *Proc. IEEE Int. Test Conf.*, 2003, pp. 423–430.



Jiang Brandon Liu (S'99–M'04) received the B.A.Sc. degree in engineering science and the M.A.Sc. degree in computer engineering from the University of Toronto, Toronto, ON, Canada, in 2000 and 2003, respectively.

He is currently with the High-Performance Tools and Methodology Group, Freescale, Austin, TX. His research interests include fault diagnosis, logic debugging, and performance optimization.



Andreas Veneris (S'96–M'99) was born in Athens, Greece. He received the Diploma in computer engineering and informatics from the University of Patras, Patras, Greece, in 1991, the M.S. degree in computer science from the University of Southern California, Los Angeles, in 1992, and the Ph.D. degree in computer science from the University of Illinois, Urbana-Champaign, in 1998.

He is currently a cross-appointed Associate Professor in the Department of Electrical and Computer Engineering and the Department of Computer Science, University of Toronto, Toronto, ON, Canada. He has coauthored one book. His research interests include computer-aided design for synthesis, diagnosis, and verification of digital circuits and combinatorics.

Prof. Veneris is a Member of the ACM, the AAAS, the Technical Chamber of Greece, and the Planetary Society. He was a co-recipient of a Best Paper Award at the 2001 Asia South Pacific Design Automation Conference.