A Deductive Technique for Diagnosis of Bridging Faults

Srikanth Venkataraman
Coordinated Science Laboratory
University of Illinois, Urbana, IL 61801

W. Kent Fuchs
School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907-1285

Abstract

A deductive technique is presented that uses voltage testing for the diagnosis of single bridging faults between two gate input or output lines and is applicable to combinational or full-scan sequential circuits. For defects in this class of faults the method is accurate by construction while making no assumptions about the logic-level wired-AND/OR behavior. A path-trace procedure starting from failing outputs deduces potential lines associated with the bridge. The information obtained from the path-trace from failing outputs is combined using an intersection graph to make further deductions. All candidate faults are implicitly represented, thereby obviating the need to enumerate faults and hence allowing the exploration of the space of all faults. Results are provided for all large ISCAS89 benchmark circuits.

1 Introduction

A *bridging fault* [1] between two lines A and B in a circuit occurs when the two lines are unintentionally shorted. When the lines A and B have different logic values, the gates driving the lines will be engaged in a drive fight (logic contention). Depending on the gates driving the lines A and B, their input values, and the resistance of the bridge, the bridged lines can have intermediate voltage values V_A and V_B (not well defined logic values of 1 or 0). This is interpreted by the logic that fans out from the bridge as shown in the shaded region in Figure 1 (a).

The logic gates downstream from the bridged nodes can have variable input logic thresholds. Thus the intermediate voltage at a bridged node may be interpreted differently by different gates. This is known as the *Byzantine Generals Problem* [2, 3] and is illustrated in Figure 1 (b). The voltage at the node A (V_A) is interpreted as a faulty value (0) by gate d and a good value (1) by gate c. Thus, different branches from a single fanout stem can have different logic values.

The feasibility of any diagnosis scheme can be evaluated using the parameters: *accuracy*, *precision*, *storage requirements* and *computational complexity*. Accurate simulation of bridging faults [4, 3] is computationally expensive. Thus, it may not be feasible to perform bridging fault simulation during diagnosis. Further, the space of all bridging faults is extremely large. For example, for the large ISCAS89 benchmark circuits, it is of the order of 10^9 faults.

Several techniques have been proposed for the diagnosis of bridging faults in combinational circuits using voltage testing. Millman, McCluskey and Acken [5] presented an approach to diagnose bridging faults using stuck-at dictionaries. Chess et al. [6], and Lavo et al. [7] improved on this technique. These techniques enumerate bridging faults and are hence constrained to using a reduced set of bridging faults extracted from the layout. Further, the construction and *storage requirements* of fault dictionaries may be prohibitive. Chakravarty and Gong [8] describe a voltage-based algorithm that uses the wired-AND (wired-OR) model and stuck-at fault dictionaries. The wired-AND and wired-OR models work only for technologies for which one logic value is always more strongly driven than the other.

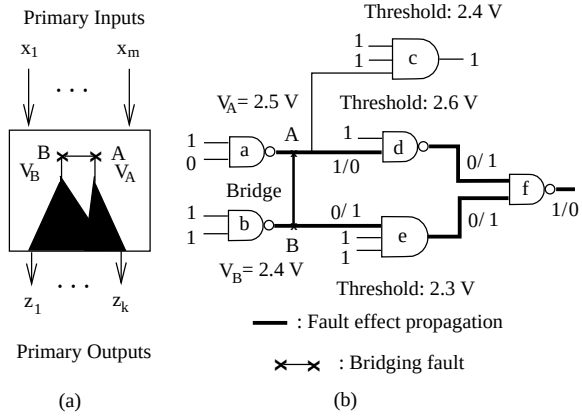


Figure 1: Bridging Fault Effect Propagation

In this paper we present a deductive technique that does not require fault dictionaries and does not explicitly simulate faults, either stuck-at or bridging. Further, no model such as wired-AND or wired-OR is assumed at the logic-level. The class of bridging faults considered are all single bridging faults between two lines in the circuit. The lines could be gate outputs, gate inputs, or primary inputs. For defects in this class of faults, the method is accurate in that the defect is guaranteed to be in the candidate list. In the following, a *failing vector* and a *failing output* refer to a vector and a primary output that fail the test on a tester (not during simulation). The deductive technique consists of two deductive procedures. The first is a path-trace procedure that starts from failing outputs and uses the logic values obtained by the logic simulation of the good circuit for each failing vector. This is used to deduce lines potentially associated with the bridging faults. The second procedure is an intersection graph constructed from the information obtained through path-tracing from failing outputs. The path-trace and the intersection graph are constructed and processed dynamically during diagnosis. The intersection graph implicitly represents all candidate bridging faults under consideration, thereby allowing pro-

This research was supported in part by Defense Advanced Research Projects Agency (DARPA) under contract DABT 63-96-C-0069, by the Semiconductor Research Corporation (SRC) under grant 95-DP-109, by the Office of Naval Research (ONR) under grant N00014-95-1-1049, and by an equipment grant from Hewlett-Packard.

cessing of the entire space of all bridging faults in an implicit manner. During diagnosis, a reduced version of the graph is maintained that retains all diagnostic information. This reduces memory usage and simulation time. Since the technique uses only logic simulation and does not explicitly simulate faults, it is fast. The technique outputs a list of candidate faults. If the resolution (the size of the candidate list) is adequate, the diagnosis is complete. Otherwise, either the candidate list can be simulated with a bridging fault simulator or other techniques [5, 6, 7, 8] can be used to improve the resolution.

2 The Path-Trace Procedure

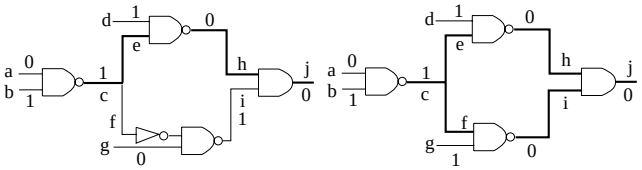
The path-trace procedure deduces lines in the circuit that are potentially associated with a bridging fault. A *potential source of error* with respect to a failing output is defined as follows.

Definition 1 *A* **potential source of error***, with respect to a failing output, is a line in the circuit from which there exists a sensitized path to that failing output on the application of the corresponding failing vector.*

Note that there is a distinction between potential sources of error and the actual source(s) of error associated with the defect. In the following, the actual source(s) of error are simply referred to as the source(s) of error. The path-trace procedure is similar to *critical path tracing* [9], and *star algorithm* [10]. However, there are some important differences. The above procedures were developed for single stuck-at faults. Hence, only one line in the circuit is assumed to be faulty. However, for bridging faults, due to the Byzantine Generals Problem, both lines could be sources of fault effects. Further, these effects may reconverge, leading to effects such as multiple-path sensitization as shown in Figure 1 (b). The voltages at lines A (V_A) and B (V_B) are both interpreted as faulty by gates d and e, and the fault effect reconverges at gate f. However, the assumption of a single bridging fault between two lines ensures that at most two lines in the circuit can be sources of error.

The logic-value of a gate input is said to be *controlling* if it determines the gate's output value regardless of other input values [11]. The path-trace procedure proceeds as follows. Start from a failing output and process the lines of the circuit in a reverse topological order up to the inputs. When a gate output is reached, observe the input values. If all inputs have noncontrolling values, continue the trace from all inputs. If one or more inputs have controlling values, continue the trace from any one controlling input. When a fanout branch is reached, continue tracing from the stem. The choice in selecting the controlling value can be exploited, as will be explained later.

We first consider the case of a single line being the source of error for a failing output, and then consider the case where both lines of a bridging fault are sources of error for a failing output. Consider a single line being the source of errors on a failing vector. When reconvergent fanout exists, the following situations could occur. In Figure 2 (a), the effects of an error from the stem c propagate to the output. However, if the paths have different parities, they will cancel each other when they reconverge. This is referred to as *self-masking* [9]. Figure 2 (b) shows an example of *multiple path sensitization* [9]. The bold lines indicate error propagation. The error from line c propagates through two paths before reconverging and propagating to an output.



(a) error from stem c propagates (b) multiple-path sensitization from c

— : Error propagation

Figure 2: Reconvergent Fanout with Single Source of Error

Lemma 1 *On any failing vector, the path-trace procedure includes all potential sources of error with respect to the failing outputs, assuming there is a single source of error.*

Proof: When the path-trace reaches a fanout branch, it continues from the stem. Hence, if the stem were a source of error, it would be included. If a gate has multiple controlling values on its inputs, then fault effects can propagate through this gate only if there exists a stem from which errors reconverge at this gate to collectively change all the controlling values. When the path-trace reaches this gate, it will continue along one of the lines having controlling values. Hence it will include the stem. □

Lemma 1 can be interpreted as follows. If the defect causes a single line to be faulty on some failing vector and this fault effect propagates to some failing output, then the path-trace includes all lines that are sensitized to that failing output. The path-trace procedure is conservative with respect to single sources of error. Not all lines in the path-trace may be potential sources of error. For example, line h in Figure 2 (b) is not a potential source of error but would be included in the path-trace. However, this conservative approach is necessary when both lines of a bridging fault could be sources of error with respect to some failing output. Note that for a single source of error, the potential sources of error are the same as critical lines [9] in the circuit. Next, we consider the case where both lines of a bridging fault are sources of error on some failing vector. If there exists at least one path between the lines of a bridging fault, then the bridging fault creates one or more feedback loops. Such a fault is referred to as a *feedback bridging fault* [11]. If no paths exist between the lines of a bridging fault, then it is called a *nonfeedback bridging fault*. A feedback bridging fault may cause *oscillations* to occur if the input vector creates a sensitized path from one line of the bridging fault to the other and this path has odd inversion parity. If such oscillations are detectable by the tester, then they can be used as additional failing outputs for the path-trace procedure. The following Lemma, Theorem and Corollary are applicable to both feedback and nonfeedback bridging faults. The symbol $A@B$ is used to represent a bridging fault.

Lemma 2 *If a bridging fault $A@B$ causes fault effect propagation to an output due to reconvergence of bridging fault effects from both lines of the bridging fault, then the path-trace procedure starting from that failing output will include at least one of the lines of the bridging fault.*

Proof: At the reconvergent gate, there exist one or more controlling input values. The path-trace continues from one of the lines with controlling input value. Thus, one of the lines of the bridging fault is covered by the path-trace. □

A case of Lemma 2 is illustrated in Figure 3. The output of gate e fails. Path-trace starts from this output and proceeds to the inputs. Since gate e has two controlling inputs, the trace continues from one of them. Node B, which is part of the bridging fault $A@B$, is covered by the path-trace.

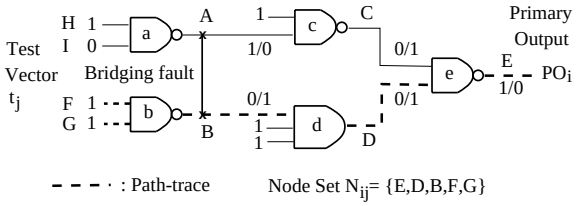


Figure 3: Path-Trace and Node Set

Definition 2 *The* **node set** N_{ij} *is defined as the set of lines that lie on the path-trace starting from failing output* PO_i *under the application of test-vector* t_j.

Theorem 1 *If neither line* A *nor line* B *of a bridging fault* $A@B$ *is in a node set* N_{ij}*, then the fault* $A@B$ *could not have caused output* PO_i *to fail under test vector* t_j.

Proof: [By contradiction] Assume that the bridging fault $A@B$ caused an output PO_i to fail on some test-vector t_j. This implies that there exists a sensitized path from A, or B, or the interaction of fault effects from both A and B, to the primary output PO_i under the application of test-vector t_j. If neither line A nor line B is in N_{ij}, then due to Lemmas 1 and 2, there exists no sensitized path to PO_i. This leads to a contradiction. □

Corollary 1 *If the defect is a single bridging fault* $A@B$*, then a node set* N_{ij} *must contain at least one of the lines* A *and* B.

Proof: Follows directly from Theorem 1. □

Note that Theorem 1 and Corollary 1 are conservative in that they make no assumptions about the resistance of the bridging fault, the gates feeding the bridging fault and their input values, and the logic input thresholds of the gates downstream from the bridging fault. The only assumption made is the presence of a single bridging fault. The information from a group of node sets can be used to make further deductions. This is performed using the concept of an *intersection graph*.

3 The Intersection Graph and Its Processing

Given a group of node sets $\{N_{ij}\}$ the intersection graph is defined as follows.

Definition 3 *The* intersection graph *is a simple undirected graph (no loops or multiple edges)* $G_I = (V_I, E_I)$ *with a vertex set* $V_I = \{N_{ij} \mid N_{ij}$ *is a node set*$\}$ *and edge* $(N_{ij}, N_{kl}) \in E$ *if* $(((i \neq k)$ *or* $(j \neq l))$ *and* $(N_{ij} \cap N_{kl} \neq \phi))$.

Figure 4 shows an intersection graph with 7 vertices. The corresponding node sets are shown within the curly brackets. The intersection graph has similar structure to the initialization graph proposed by Chakravarty and Gong [8]. However, there are important differences. The initialization graph is constructed using only structural information while the intersection graph is constructed using logic information exploited by the path-trace procedure. The initialization graph is created statically once before diagnosis and processed. However, the intersection graph is updated and processed dynamically during diagnosis. A reduction procedure maintains a reduced version of the graph without losing diagnostic information. The intersection graph has interesting structural properties that are useful for performing deduction and for maintaining reduced graphs to help reduce memory requirements and simulation time.

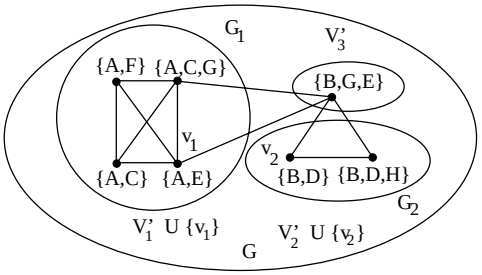


Figure 4: Intersection Graph and Its Properties

3.1 Structural Properties

Property 1 *If* G_I *has two vertices such that* $(v_1, v_2) \notin E_I$*. The set of vertices* $V_I^{'} = V_I - \{v_1, v_2\}$ *can be partitioned into three sets* $V_1^{'}, V_2^{'}$ *and* $V_3^{'}$ *such that* $\forall v_1^{'} \in V_1^{'}, (v_1^{'}, v_1) \in E_I$ *and* $(v_1^{'}, v_2) \notin E_I$*;* $\forall v_2^{'} \in V_2^{'}, (v_2^{'}, v_1) \notin E_I$ *and* $(v_2^{'}, v_2) \in E_I$*;* $\forall v_3^{'} \in V_3^{'}$*,* $(v_3^{'}, v_1) \in E_I$ *and* $(v_3^{'}, v_2) \in E_I$.

Proof: Let N_{ij}^1 and N_{ij}^2 be the node sets corresponding to v_1 and v_2. From Corollary 1, N_{ij}^1 and N_{ij}^2 contain at least one of the lines of the bridging fault $A@B$. Since $(v_1, v_2) \notin E_I$, N_{ij}^1 contains only one of the lines A and B (say A). This implies that N_{ij}^2 contains the other line (B). Consider any arbitrary vertex $v_3 \in V_I^{'} = V_I - \{v_1, v_2\}$. From Corollary 1 it follows that the node set corresponding to v_3 contains at least one of the lines A and B. Thus v_3 is adjacent to at least one of v_1 and v_2. This implies that one of the following three conditions holds: v_3 is adjacent to v_1 and not adjacent to v_2; v_3 is adjacent to v_2 and not adjacent to v_1; v_3 is adjacent to both v_1 and v_2. □

Property 2 *If* $V_1^{'}, V_2^{'}$*, and* $V_3^{'}$ *are the three sets obtained by Property* 1 *when* $(v_1, v_2) \notin E_I$*, then* $V_1^{'} \cup \{v_1\}$ *and* $V_2^{'} \cup \{v_2\}$ *are cliques.*

Proof: From Corollary 1 and Property 1 it follows that the node sets corresponding to every $v_i \in V_1^{'} \cup \{v_1\}$ contain one and only one of A and B (say A), while the node sets corresponding to every $v_j \in V_2^{'} \cup \{v_2\}$ contain the other line (B). Thus $V_1^{'} \cup \{v_1\}$ and $V_2^{'} \cup \{v_2\}$ are cliques. □

Figure 4 illustrates these properties. The intersection graphs can be reduced while maintaining their properties. This reduces the number of vertices and edges. Further, this also reduces the number of node sets that need to be maintained and their sizes. Thus the reduction process, which is done dynamically during diagnosis, can help reduce memory and simulation time. The following Corollary, which follows from Property 2, is used in the reduction process.

Corollary 2 *If the intersection graph is not a clique, and* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$ *are the subgraphs induced by* $V_1^{'} \cup \{v_1\}$ *and* $V_2^{'} \cup \{v_2\}$ *respectively using Property* 1*, then all node sets in* V_1 *contain only one of the lines* A *or* B *of the bridge, while all the node sets in* V_2 *contain the other line.*

3.2 Intersection Graph Processing

Corollary 2 is used by the procedure shown in Figure 5 to reduce the intersection graph. An irreducible intersection graph is either a complete graph or has the following characteristic. For each $(v_i, v_j) \notin E_I$, the $V_1^{'}$ and $V_2^{'}$ sets are empty. An example of the reduction procedure is shown in Figure 6. The initial intersection

1 : $G_I = (V_I, E_I)$: The intersection graph
2 : $V(N_{ij})$: Vertex corresponding to node set N_{ij}
3 : **Procedure Reduce_Intersection_Graph(G_I)**
4 : **while** (!irreducible)
5 : **if** (!$clique(G_I)$)
6 : Let $(v_1, v_2) \notin E_I$
7 : // Possible to find v_1, v_2 since G_I is not a clique
8 : $V_1^{'} = \{v_i \mid (v_i, v_1) \in E_I$ and $(v_i, v_2) \notin E_I\}$
9 : $V_2^{'} = \{v_j \mid (v_j, v_2) \in E_I$ and $(v_j, v_1) \notin E_I\}$
10 : // By Property 2 and Corollary 2
11 : **if** ($V_1^{'} \neq \phi$)
12 : $N^1_{reduced} = \cap N_{ij}\ \forall V(N_{ij}) \in V_1^{'} \cup \{v_1\}$
13 : **if** ($V_2^{'} \neq \phi$)
14 : $N^2_{reduced} = \cap N_{ij}\ \forall V(N_{ij}) \in V_2^{'} \cup \{v_2\}$
15 : Replace $v_i \in V_1^{'} \cup \{v_1\}$ with $V(N^1_{reduced}) = v_1^{'}$
16 : Replace $v_j \in V_2^{'} \cup \{v_2\}$ with $V(N^2_{reduced}) = v_2^{'}$
17 : Recompute edges incident on $v_1^{'}$ and $v_2^{'}$

Figure 5: Procedure for Reducing the Intersection Graph

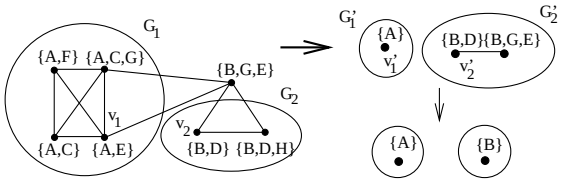
graph is reduced two times to obtain an irreducible graph with two disjoint vertices. The dynamic processing of G_I proceeds as follows. After each node set N_{ij} is obtained, update G_I. Reduce the intersection graph until an irreducible graph is obtained. After all node sets are processed, the irreducible intersection graph obtained contains the candidate bridging faults. The candidate list (C) is obtained from the irreducible graph (G_{IR}) using the following rules.

1. If G_{IR} has two disconnected components, each of which has one vertex, then let N^1_{ij} and N^2_{ij} be the node sets associated with the two vertices. $C = \{A@B \mid A \in N^1_{ij}$ and $B \in N^2_{ij}\}$.
2. If G_{IR} has one component that is not a complete graph, then for each $(v_1, v_2) \notin E$ let N^1_{ij} and N^2_{ij} be the node sets associated with v_1 and v_2. $C_k = \{A@B \mid A \in N^1_{ij}$ and $B \in N^2_{ij}\}$. $C = \cap_{\forall k} C_k$.
3. If G_{IR} is a complete graph, then let $|N_{ij}| \leq |N_{kl}| \forall N_{ij}, N_{kl} \in V$. $C = \{A@B \mid A \in N_{ij}\}$.

The reduced intersection graph is a compact way to implicitly represent the space of candidate bridging faults. Further, the reduction procedure prunes the space of candidate bridging faults without losing diagnostic information. The defect is guaranteed to be in the candidate list by construction. The candidate list will include other faults which are logically equivalent or diagnostically equivalent with respect to the test set. A better test set may distinguish between some of these faults, thereby increasing the diagnostic resolution. When G_{IR} is a complete graph, only one of the lines of the bridge can be determined with certainty. This results in a *partial diagnosis*. The experimental results indicate that partial diagnosis does not occur often.

3.3 Implementation Issues and Complexity

The major operation performed during G_I processing is its reduction. The basic operation needed by the reduction procedure is set intersection. Further, the node sets need to be stored for each vertex



(a) Initial intersection graph (b) Reduced intersection graph

Figure 6: An Example of the Intersection Graph Reduction

of G_I. The node sets are represented as bit-vectors with a value of 1 indicating the presence of a node and a 0 indicating the absence of one. If there are n lines in the circuit, the size of a node set is $\lceil n/8 \rceil$ bytes. The bit vector representation allows for efficient set intersection using the bitwise AND operator. As a result of the dynamic processing of G_I, its size grows and shrinks. Hence, the data structure chosen to represent G_I is a two-dimensional linked list. G_I has $|V_I|$ vertices. Assuming 4 bytes each for pointers and vertex indices, the worst-case memory requirement for G_I and its associated node sets is $(|V_I| \times \lceil n/8 \rceil + 8|V_I| + 8|V_I|^2)$ bytes. Since $|V_I|$ is typically much smaller than n, the worst-case space complexity is $O(|V_I| \times n)$. The worst-case size of $|V_I|$ is n_{fail}, where n_{fail} is the total number of failing outputs on all failing vectors. The reduction procedure results in $|V_I|$ being much smaller than n_{fail}, thereby reducing the memory requirements.

The reduction procedure computes the $V_1^{'}$ and $V_2^{'}$ sets based on Corollary 2 by exploring the edges of $\overline{E_I}$. Typically, $|\overline{E_I}|$ is small. For each edge in $\overline{E_I}$ the reduction procedure computes $V_1^{'}$ and $V_2^{'}$. Each intersection operation between the node sets of two vertices of G_I reduces the number of vertices of G_I by 1. Thus the maximum number of intersections possible in the procedure $reduce\_intersection\_graph()$ is $|V_I| - 2$. Thus the worst-case time complexity for the procedure $reduce\_intersection\_graph()$ is $O(|V_I|)$ intersections. Here again, the reduction procedure results in small $|V_I|$ values, thereby reducing the simulation time.

4 Heuristics to Improve Resolution

When the path-tracing procedure reaches a gate with multiple controlling inputs, one of them is chosen. The choice of input impacts the size of the resultant node set, its elements, and hence, impacts the diagnostic resolution. The smaller the size of the node set, the smaller is the intersection with other node sets, and the greater is the likelihood of reducing the intersection graph. Two conditions are checked to select the controlling input in such a manner that the size of the resultant node set is reduced. The first is based on *fanout*. When the path-trace reaches a stem, it continues from the stem unconditionally. When a controlling input is the branch of a stem, one of whose other branches has been chosen, then this input should be selected, since the stem has to be selected anyway [10]. The second condition involves checking the *controllability* of the line. SCOAP controllability measures are used. The most easily controllable input (check for 0-controllability if the logic value of the line is 0 and vice-versa) is likely to give the smallest node set. If the same gate is reached in two different applications of the path-trace and the same choice of controlling inputs exists, then selecting different inputs for the two runs can potentially result in a smaller intersection between the two resultant node sets. A *dirty bit* is set when the path-trace chooses a controlling input. This input

is avoided in future invocations of the path-trace procedure.
Based on the above conditions, three heuristics are defined below. *Heuristic 1* chooses controlling input randomly. *Heuristic 2* chooses controlling input by checking for fanout followed by controllability. *Heuristic 3* chooses controlling input by checking for dirty bit followed by fanout and then controllability. The overall diagnosis procedure is shown in Figure 7.

```
1 :  for each test vector t_i
2 :      if faulty outputs
3 :          Perform good circuit simulation with t_i
4 :          for each failing output PO_j
5 :              N_ij ← path-trace from failing output PO_j
6 :              update_intersection_graph(G_I, N_ij)
7 :              reduce_intersection_graph(G_I)
8 :  output_candidate_list(G_I)
```

Figure 7: The Diagnosis Procedure

5 Experimental Results

The diagnosis procedure was implemented in $C++$. All experiments were performed on a SUN SPARCStation 20 with 64MB of memory for the full-scan versions of the ISCAS89 sequential benchmark circuits [12]. In practice, the failing responses used as input for the diagnosis procedure would be obtained by testing the failing circuit on a tester. For our diagnosis experiments, the failing responses were generated using the accurate bridging fault simulator E-PROOFS [4] to ensure that the diagnostic experiments were as realistic as possible. The cell libraries for the circuits were generated manually [4]. The test vectors used were compact tests generated to target stuck-at faults [13]. Ideally, diagnostic test sets for bridging faults would be the best choice. All large ISCAS89 benchmark circuits were considered.

For each of the benchmark circuits, a random sample of 500 single two-line bridging faults were injected one at a time. For each one of these faults, the failing responses were obtained by performing bridging fault simulation on the given test set using E-PROOFS. Faults that do not produce any failing outputs were dropped. For the rest of the faults, the failing responses were used to perform diagnosis. The diagnosis results are summarized in Tables 1 and 2. The average, minimum and maximum sizes of the candidate lists are shown in Table 1 for the three different heuristics. The average size of the candidate list is a few hundred faults, which is a significant reduction from the space of all faults. Further, as expected, heuristics 2 and 3 improve the diagnostic resolution over heuristic 1. The reduction can be significant. For example, for s38584, the average size of the candidate list is reduced by a factor of 4. Note that in some cases, the method uniquely identifies the fault (resolution of 1). The best resolutions are indicated in bold.

The average sizes of the node sets and the intersection graph are shown in Figure 2. As expected, heuristic 2 does the best in terms of node set sizes. Both heuristic 2 and heuristic 3 do better than heuristic 1 in terms of the average size of the intersection graph. The average values of the execution time, number of failing outputs and percentage of faults that were partially diagnosed is given for heuristic 2. Other interesting observations can be made from Table 2. Note that the average size of the sets is very small and appears to be independent of the circuit size. Further, it is about 2–3 orders of magnitude smaller than the total number of lines in the circuit, thereby suggesting that the path-trace procedure is efficient. The average size of the intersection graph ($|V_I|$) is about a quarter of the total number of failing outputs, indicating that the graph reduction procedure is useful. As expected heuristic 2 does the best in terms of the average size of the node set and heuristic 3 does the best in terms of the average size of the intersection graph ($|V_I|$).

Note that the procedure is accurate by construction; that is, the defect is guaranteed to be in the candidate list. The distribution of the sizes of the candidate lists is shown in Figure 8 for s13207 and s38417. This trend is observed for the other circuits as well. For about 10% of the faults, the resolution is adequate (less than 20 candidates) to consider the diagnosis complete. For about 80% of the faults, the resolution is such (between $20 - 500$ faults) that the candidate list is small enough to be accurately simulated using a bridging fault simulator as a post-processing step. In about 25% of the cases, the diagnosis is partial; that is, only one of the lines of the bridge can be determined with certainty. In such cases, and if the resolution is so large that bridging fault simulation cannot be performed, then the diagnosis procedure can be followed with other techniques [5, 6, 7, 8] using the candidate list to improve the resolution. Note that these resolutions were obtained using a compacted stuck-at test set. We expect that there would be better resolution with better test sets.

Table 1: Diagnostic Resolution

Circuit	Candidate List Size								
	Heuristic 1			Heuristic 2			Heuristic 3		
	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.	Max.
s1196f	125.2	1	1656	**60.7**	1	828	76.7	1	765
s1238f	123.0	1	1080	**62.7**	1	722	76.3	1	1092
s1423f	98.2	1	2303	**80.9**	1	3526	83.9	1	3526
s1488f	70.7	8	165	**65.7**	2	280	68.9	2	375
s1494f	74.5	4	120	**68.9**	2	140	71.8	2	140
s5378f	284.2	2	3243	203.2	1	1659	**194.2**	2	1386
s9234f	866.9	18	7848	**771.4**	12	7020	809.4	12	7020
s13207f	240.3	1	4199	167.8	1	3809	**160.5**	1	1328
s15850f	694.4	1	10206	679.4	1	9100	**557.3**	1	9178
s35932f	128.6	4	690	125.6	4	630	**115.4**	4	630
s38417f	280.4	1	2793	240.1	1	2842	**219.3**	1	2475
s38584f	743.5	48	2130	**191.6**	1	2288	204.2	1	2600

The diagnosis procedure requires very small execution times, as seen in column 8 of Table 2. The procedure requires only the logic simulation of failing vectors and the path-trace procedure from failing outputs. Both of these procedures are linear in the size of the circuit. Further, the graph reduction procedure is linear in the size of its vertex set ($|V_I|$). Techniques such as those used in [5, 6, 7, 8] require either the storage of stuck-at fault dictionaries or the simulation of stuck-at faults during diagnosis. As seen in columns 12 and 13 of Table 2, the storage requirements for dictionaries can be very large, and the simulation time is about an order of magnitude larger than that required for the diagnosis procedure. This is expected since fault simulation has greater than linear complexity in the size of the circuit. Further, fault simulation without fault dropping needs to be performed. Techniques such as those used in [5, 6, 7] also need to enumerate bridging faults and are hence constrained to use a small set of realistic faults. This trade-off between resolution and complexity suggests that our diagnosis procedure, which is both space- and time-efficient, could be attempted first, and then be complemented by other procedures if greater resolution is required.

Table 2: Diagnosis Results and Comparison with Techniques Using Stuck-at Fault Information

Circuit	Average Size of Node Set			Average $\lvert V_I \rvert$			Average Values (Heuristic 2)			Stuck-at fault Information		
	Heu.1	Heu.2	Heu.3	Heu.1	Heu.2	Heu.3	Exec. Time (s)	# Fail POs	Partial Diag. (%)	# of Faults	Storage † (Bytes)	Exec. Time ‡ Fault Sim. (s)
s1196f	32.3	**22.7**	25.3	23.6	20.5	**19.9**	0.28	71.8	0.24	1242	0.57 M	4.25
s1238f	30.6	**22.7**	24.9	27.5	25.2	**23.6**	0.35	77.3	0.25	1355	0.67 M	4.63
s1423f	28.8	25.1	**24.9**	7.8	**6.9**	7.1	0.11	26.1	0.12	1515	0.38 M	1.51
s1488f	35.6	**29.2**	30.6	20.6	20.6	**19.5**	0.23	42.8	0.28	1486	0.47 M	4.13
s1494f	29.5	**20.9**	25.2	14.9	11.4	**10.7**	0.20	27.5	0.24	1506	0.47 M	4.08
s5378f	52.9	**48.7**	49.5	13.9	13.4	**13.2**	1.32	66.5	0.22	4603	24.6 M	16.01
s9234f	40.2	**36.3**	37.0	18.7	19.3	**19.3**	2.32	65.4	0.35	6927	47.1 M	34.61
s13207f	29.6	28.2	**27.4**	62.5	58.2	**27.4**	31.1	298.3	0.16	12311	0.51 G	131.55
s15850f	68.8	**62.5**	66.3	21.6	**22.0**	24.6	7.43	92.3	0.28	11725	0.17 G	62.70
s35932f	20.1	18.4	**17.6**	3.7	3.6	**3.6**	6.4	8.5	0.34	46006	0.24 G	35.71
s38417f	46.4	**38.2**	38.9	11.6	11.5	**11.5**	25.0	61.6	0.14	31180	1.13 G	151.30
s38584f	38.9	**28.7**	29.4	31.3	**28.7**	29.4	30.4	64.9	0.38	36303	1.56 G	214.51

† Full fault dictionary in matrix format

‡ w/o fault dropping

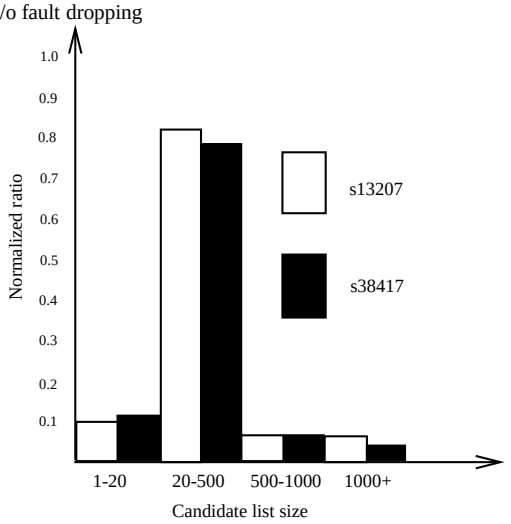


Figure 8: Distribution of Candidate List Size

6 Conclusions and Future Work

A deductive procedure for the diagnosis of bridging faults, which is accurate and experimentally shown to be both space- and time-efficient, has been described. The information obtained from a path-trace procedure from failing outputs is combined using an intersection graph, which is constructed and processed dynamically, to make the deduction. The intersection graph provides an implicit means of representing and processing the space of candidate bridging faults without using dictionaries or explicit fault simulation. The procedure assumes a single bridging fault between two lines. If the defect involves multiple faults or shorts between multiple lines, then the properties of G_I may be violated. Extensions to multiple faults or shorts between multiple lines require looking for larger sized cliques (K_n, $n \geq 3$) in the graph $\overline{G_I}$. An interesting application of this work is in the area of design error location. For design errors of multiplicity 2, the diagnosis procedure can be used without any modification. Higher multiplicity errors require extensions.

References

[1] K. C. Y. Mei, "Bridging and Stuck-at Faults," *IEEE Tran. on Computers*, vol. C-23(7), pp. 720–727, July 1974.

[2] J. M. Acken and S. D. Millman, "Fault Model Evolution for Diagnosis: Accuracy vs. Precision," in *Proc. of the Custom Integrated Circuits Conf.*, pp. 13.4.1–13.4.4, 1992.

[3] P. C. Maxwell and R. C. Aitken, "Biased Voting: A Method for Simulating CMOS Bridging Faults in the Presence of Variable Gate Logic Thresholds," in *Proc. of the IEEE Intl. Test Conf.*, pp. 63–72, Oct. 1993.

[4] G. S. Greenstein and J. H. Patel, "E-PROOFS: A CMOS Bridging Fault Simulator," in *Proc. of the IEEE/ACM Intl. Conf. on Computer-Aided Design*, pp. 268–271, Nov. 1992.

[5] S. D. Millman, E. J. McCluskey, and J. M. Acken, "Diagnosing CMOS Bridging Faults with Stuck-at Fault Dictionaries," in *Proc. of the IEEE Intl. Test Conf.*, pp. 860–870, Oct. 1990.

[6] B. Chess, D. B. Lavo, F. J. Ferguson, and T. Larrabee, "Diagnosing of Realistic Bridging Faults with Stuck-at Information," in *Proc. of the IEEE/ACM Intl. Conf. on Computer-Aided Design*, pp. 185–192, Nov. 1995.

[7] D. B. Lavo, T. Larrabee, and B. Chess, "Beyond the Byzantine Generals: Unexpected Behavior and Bridging Faults Diagnosis," in *Proc. of the IEEE Intl. Test Conf.*, pp. 611–619, Oct. 1996.

[8] S. Chakravarty and Y. Gong, "An Algorithm for Diagnosing Two-Line Bridging Faults in CMOS Combinational Circuits," in *Proc. of the Design Automation Conf.*, pp. 520–524, June 1993.

[9] P. R. Menon, Y. Levendel, and M. Abramovici, "SCRIPT: A Critical Path Tracing Algorithm for Synchronous Sequential Circuits," *IEEE Tran. on Computer Aided Design*, vol. 10, pp. 738–747, June 1991.

[10] S. B. Akers, B. Krishnamurthy, S. Park, and A. Swaminathan, "Why is Less Information From Logic Simulation More Useful in Fault Simulation?," in *Proc. of the IEEE Intl. Test Conf.*, pp. 786–800, Sept. 1990.

[11] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital System Testing and Testable Design*. AT&T Bell Laboratories and W. H. Freeman and Company, 1990.

[12] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *IEEE Intl. Symp. on Circuits and Systems*, pp. 1929–1934, May 1989.

[13] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits," *IEEE Tran. on Computer Aided Design*, vol. 14, pp. 1496–1504, December 1995.