JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 19, 571-587 (2003)

# Efficient Double Fault Diagnosis for CMOS Logic Circuits With a Specific Application to Generic Bridging Faults

HONG-CHOU KAO, MING-FU TSAI, SHI-YU HUANG, CHENG-WEN WU,
WEN-FENG CHANG[*] AND SHYUE-KUNG LU[+]
*Department of Electrical Engineering*
*National Tsing-Hua University*
*Hsinchu, 300 Taiwan*
[*]*Department of Computer Science and Information Engineering*
*Van-Nung Institute of Technology*
*Taoyuan, 320 Taiwan*
[+]*Department of Electrical Engineering*
*Fu-Jen Catholic University*
*Hsinchuang, 242 Taiwan*

Fault diagnosis that predicts the most likely fault sites in a faulty chip is an important step for manufacturing yield improvement or design debugging. In this paper, we address the problem of double fault diagnosis for full-scan designs. Our algorithm aims to identify both faults accurately. The features of our algorithm include the following. (1) The proposed algorithm is not limited to any particular fault type. (2) An effective selection heuristic is incorporated to significantly reduce the diagnosis time, while retaining a high success rate of catching faults. (3) The inject-and-evaluate paradigm proposed in [6] is incorporated to accurately screen out unlikely fault candidates. Experimental results on ISCAS85 benchmark circuits injected with a generic bridging fault, two stuck-at faults, or two gate-type faults show that both faults can be caught simultaneously within several minutes with a high success rate.

***Keywords:*** VLSI testing, fault diagnosis, logic diagnosis, fault model, fault simulation

## 1. INTRODUCTION

Fault diagnosis involves the identification of the fault locations in an IC that fails testing. Based on the identified fault information, the manufacturing process can then be adjusted accordingly to improve the manufacturing yield [9, 11, 12]. Most papers about diagnosis assume that there is only one fault affecting every faulty output, known as the single fault assumption. With the increasing density of ICs, the single fault assumption may not always be valid. Therefore, double fault diagnosis is becoming more and more necessary. The theoretical foundation for double fault diagnosis has been established in [6]. However, the process is too time-consuming. In this paper, we focus on reducing the time for finding double faults that affect certain faulty primary outputs simultaneously in a full-scan design.

There are two major types of fault diagnosis methods: cause-effect analysis [12], and effect-cause analysis [2, 5, 6, 10]. A detailed comparison of these algorithms can be

Received July 31, 2001; revised July 8, 2002; accepted August 8, 2002.
Communicated by Jiun-Lang Huang.

found in [1]. The algorithm to be proposed belongs to effect-cause analysis, i.e., we do not build any memory-intensive fault dictionary like the cause-effect approaches. The fault effects at the primary outputs (the syndromes) are directly analyzed, and then a procedure is used to identify the potential fault sites (the causes of the chip failure).

One important issue for diagnosis is how to model the faults. Traditionally, stuck-at or bridging faults are used to assume the behavior of the faults, leading to inaccurate diagnosis for certain cases [3]. To overcome this limitation, we do not use such an assumption in our approach. The experimental results show that our approach can indeed handle many types of faults.

Another issue to be dealt with is the so-called candidate space explosion problem. Since a double fault is associated with a signal pair ($w_1$, $w_2$), a circuit with $n$ signals will yield $O(n^2)$ possible *double fault candidates*. An algorithm explicitly checking every candidate signal pair would be too time-consuming. To address this issue, we propose a filtering heuristic that eliminates less likely signal pairs from the $O(n^2)$ candidate space. The heuristic aims to reduce the candidate space from $O(n^2)$ to $O(n)$ while minimizing the number of real faults that could be mistakenly eliminated. The quality of such a filtering heuristic can be measured by the success rate defined below.

**Definition 1:** (*Success rate*). A filtering heuristic takes the set of all possible double fault candidates (signal pairs) as the input and produces a *reduced* set of double fault candidates. The success rate of such a filtering heuristic is defined as the rate of successfully containing the real fault pair in the reduced set. In other words, it is the rate of not overlooking the real fault pair during the candidate filtering process.

It has been observed that the bridging fault (that unintentionally connects two signals) is one of the most common faults in a CMOS logic circuit. A diagnostic algorithm based on the single fault assumption is not able to diagnose a bridging fault effectively because a bridging fault involves two signals. For such a fault, we need a generic double fault diagnostic algorithm.

Traditionally, AND-bridging or OR-bridging effects are assumed when two signals are shorted together. Here, we follow the convention that a fault affects two signals, but not more. Since neither AND-bridging nor OR-bridging accurately reflects the true behavior of a bridging fault in a CMOS logic circuit, a simple model could lead to misleading results. To overcome this limitation, we propose a generic bridging fault model in which both signals involved in a bridging fault can take on a logical value of either '0' or '1' independently. The details of such a generic bridging fault is discussed in section 4.

The main contribution of this paper is an efficient double fault diagnosis flow. It has several merits. Firstly, it is independent of the fault types, and thus, more applicable to real-world cases. Secondly, it is efficient in terms of CPU time because of the incorporation of a filtering heuristic that efficiently cuts down on the candidate space. Thirdly, it pin-points the fault sites accurately by fault simulation [6]. The experimental results of ISCAS85 benchmark circuits demonstrate that it can be successfully applied to many types of faults, including the proposed generic bridging fault. On the average, the success rate of catching the faults is 96.3%, while the CPU time is on the order of minutes.

The rest of the paper is organized as follows. Section 2 provides the preliminaries.

Section 3 describes the overall diagnosis flow and the filtering heuristic. Section 4 discusses the generic bridging fault model. Section 5 presents the experimental results. Section 6 concludes.

## 2. PRELIMINARIES

In this paper, the faulty Chip Under Diagnosis is referred to as CUD. It is assumed that the design is implemented with the full-scan methodology, and its functionality is represented as a combinational gate-level netlist, which is denoted as the MODEL in this paper. The primary input (PI) signals are denoted as $\{x_1, x_2, ..., x_n\}$. The primary output (PO) signals of the CUD and the MODEL are denoted as $\{z_1^C, z_2^C, ..., z_m^C\}$ and $\{z_1^M, z_2^M, ..., z_m^M\}$, respectively, where $m$ is the total number of primary outputs. We assume that the set of input test vectors, denoted as T = $\{v_1, v_2, ..., v_k\}$, has been generated in advance.

**Definition 2:** (*Output pair*) $(z_i^C, z_i^M)$ is called an output pair for every $i$, $1 \leqq i \leqq m$.

**Definition 3:** (*Mismatched output*) An output pair $(z_i^C, z_i^M)$ is called *mismatched* if there exists an input test vector $v_i$ such that $v_i$, when applied to both the CUD and the MODEL, produces different binary values at $z_i^C$ and $z_i^M$. In particular, we call the primary output $z_i^M$ in the MODEL a *mismatched output*.

**Example 1:** In Fig. 1, the input vector $v$ when applied to both the CUD and the MODEL, produces mismatches at the first and fifth output pairs.

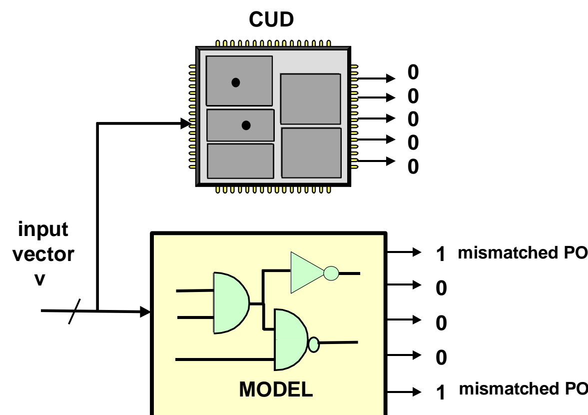


Fig. 1. Illustration of the mismatched output.

**Definition 4:** (*Resolving a mismatched output*) In the sequel, resolving a mismatched output $z_i^M$ in the MODEL means that the response of $z_i^M$ can be changed to be equivalent to that of its partner $z_i^C$ after injecting binary values to certain signals in the MODEL.

**Definiton 5:** (*Failing input vector*) An input vector is called a *failing input vector* if it creates a mismatch at any output pair.

## 3. THE ALGORITHM

The outline of the overall algorithm is shown in Fig. 2. The inputs include the MODEL, the faulty response (also called *syndromes*) of the CUD, and the set of failing input vectors. The entire algorithm has four phases.

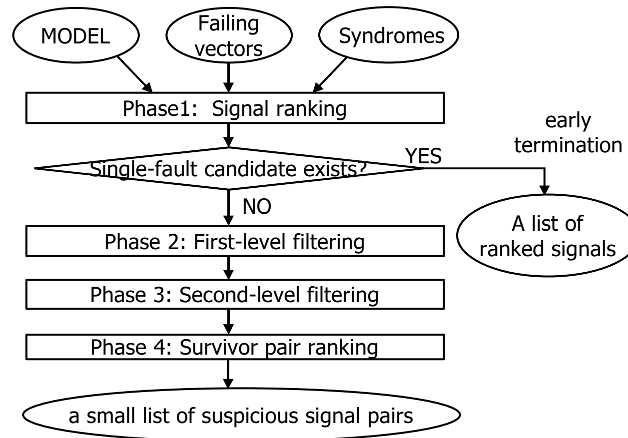


Fig. 2. The overall diagnosis flow.

(1) Phase 1: Rank the suspicious degree of each signal.
(2) Phase 2: Construct the initial set of double fault candidates. Each candidate corresponds to a signal pair. Use a simple filtering condition to rule out unlikely candidates *quickly* to speed up the subsequent process.
(3) Phase 3: Perform stringent examination on the candidates produced after Phase 2. A mall set of *survivor candidates* (*signal pairs*) is produced.
(4) Phase 4: *Rank the survivor candidates*, aiming to catch both faulty signals simultaneously.

**Phase 1: Signal Ranking**

In the first phase, signals are sorted by an improved ranking heuristic, which will produce a rank for each signal. This rank indicates a signal's possibility of being faulty. The quality of such a heuristic is often measured by its *first-hit index*, which is the index of the first signal in the ranked list that turns out to be a true faulty signal.

**Example 2:** (First-hit index example) Suppose a CUD has ten signals, {$a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$, $w_1$, $w_2$}, and there is a bridging fault at ($w_1$, $w_2$). Let the sorted signal list after the ranking heuristic be: ($a$, $w_2$, $b$, $c$, $d$, $e$, $f$, $g$, $w_1$, $h$). Then, the real faulty signal with the highest rank is $w_2$, and thus, the first-hit index is 2. The improved signal ranking heuristic

is the combination of two ideas: curability vector [7] and reward and penalty [10]. The pseudo-code is shown in Fig. 3.

```
ranking-heuristic(MODEL, FV, syndromes)
/* FV denotes the set of failing input vectors */
{
   initialize each signal's rank1 and rank2 to 0;
   for each failing input vector v {
         perform logic simulation;
         for each signal f {
            flip the value at f ;
            run a modified fault simulation;
            if ( every output is now a match ) // A curable vector
               cv(f) = cv(f) + 1;
             }
             score (f) = score (f) + curable_outputs(f, v)
                         - 0.5*new_mismatched_outputs(f, v);
         }
   }
   sort the signals using cv as the first key and
   score as the second key.
}
```

Fig. 3. The signal ranking heuristic in Phase 1.

**Definition 6:** (*Curable measures*) For a given failing input vector $v$ and a signal $f$ in the MODEL, flipping the value at $f$ may cause certain outputs of the MODEL to change values, too. If an originally mismatched output in the MODEL becomes a match (with its partner in the faulty chip), then it is called a *curable output*. In other words, the mismatch at this output can be cured by an injection at $f$. So, $f$ is suspected of being faulty because changing its value could affect the mismatched output. If an originally matched output becomes a mismatch after such an injection, then it is called a *new mismatched* output. If every output becomes a match, then the failing input $v$ is called a *curable vector* of signal $f$.

The ranking algorithm uses two variables – *cv* and *score* – for each signal to keep track of its suspicious degree. The first variable, *cv*, records a signal's total number of curable vectors while the second variable, *score*, implements the concept of "*reward and penalty*". The reward corresponds to the number of curable outputs, while the penalty corresponds to the number of new mismatched outputs times weighting factor 0.5. This weighting factor 0.5 was determined through experiments on ISCAS85 benchmark circuits. The thinking behind such a scoring rule is that a signal with a larger number of curable outputs is considered to be more suspect. But if flipping its value also creates

new mismatched outputs, then its degree of suspection is reduced. At the end of the two-level loops, the signals are sorted based on their cumulative *cv* and *score* values. The signals with a larger *cv* value will be assigned a higher rank. For signals with the same *cv* (i.e., the same number of curable vectors), the values of *score* are then used as the tie-breaker.

**Example 3:** (Computation of a signal's ranks) Consider the MODEL in Fig. 4. The input vector (x1, x2, x3, x4) is (0, 1, 1, 0). Assume that the response of the CUD to this vector is (0, 1), while the response of the MODEL is (1, 1) as illustrated. As a result, there is a mismatch at the first output pair. Now let us compute the ranks of the signal *f* in the MODEL. The value at this signal is changed from 1 to 0. This effect ripples through the circuit and changes both output values from 1 to 0. Therefore, the mismatch at the first output is cured. However, the original matched output now becomes a new mismatched output. As a result, we know that this input vector is not curable by signal *f*, and rank1 is not changed. Also, the rank2 due to this vector is (reward – 0.5 · penalty) = (1 – 0.5 · 1) = 0.5.

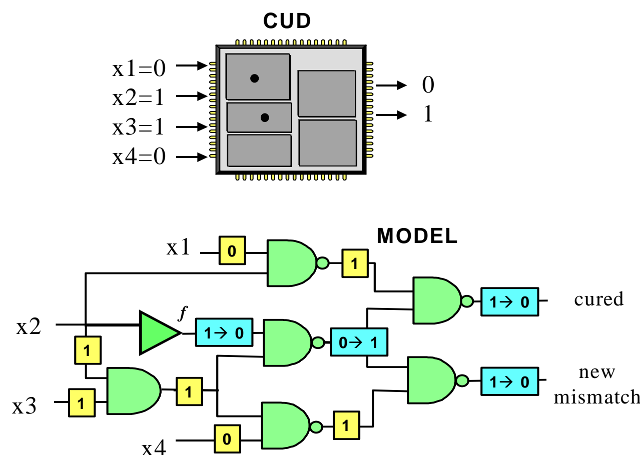


Fig. 4. Example of the computation of a signal's ranks.

It is worth mentioning that this algorithm is actually an improvement over the one reported in [7] (as will be shown in the experimental results). It emphasizes that the curable-vector-based approach can co-exist with the reward-and-penalty-based approach, while achieving an even better result.

The implementation of the modified fault simulation affects the efficiency of this algorithm significantly. An event-driven type of fault simulation often leads to a significant speed up over the compiled-code fault simulation, especially when the circuit is large. In the following, we describe the procedure of processing one failing input vector *v* using the event-driven fault simulation. First of all, a fault-free simulation is performed on the MODEL and the value of each signal is recorded. Next, a loop iterates through each candidate signal *f*. For a candidate signal, the following steps are performed (assuming that the fault-free value is denoted as $f(v)$):

■ Step 1: Set the *initial event queue* to have only one element, ($f$, $f(v)$').Note that an event corresponds to a signal-value pair. The key of an event is given by the topological level from the primary inputs of the signal in the signal-value pair. An event queue is the most important data structure in this event-driven fault simulation process, which can be implemented as an array with the min-heap structure. In other words, the *first* element of the array is always the event with the *smallest key*. Whenever an element (i.e., an event) is inserted into this queue, the array needs to be adjusted to maintain its min-heap property. This property makes the subsequent event retrieval easier.

■ Step 2: Perform an *event-processing loop* until there are no events left in the event queue. The body in this loop consists of a number of subtasks. First, the event of the minimum key (i.e., the first element in the array implementing the event queue) is retrieved. Let the signal in this retrieved event be $f_{event}$. Then, the logic gates driven by signal $f_{event}$ are re-evaluated. If any of these logic gates results in a value different from its fault-free value after the re-evaluation, then a *new event* is created. Every new event is inserted into the queue and the queue is adjusted to satisfy the min-heap property. Such an adjustment has a time complexity of O(log |queue-size|). The reason that we always retrieve the event of minimum key for processing is to guarantee that the simulation is done following the topological order in which logic gates closer to the primary inputs are always processed before any gate in their fanout region.

■ Step 3: Update the total numbers of curable vectors and curable outputs of the candidate signal $f$. When the event queue is exhausted, the computation of the final values at the outputs of the MODEL due to the flipping of a signal's value is completed. We then check each output pair. If each output pair is now equivalent, then the failing input vector $v$ is claimed as a curable vector of signal $f$. For an output pair ($z_i^C$, $z_i^M$), if the value of $z_i^M$ before the injection and the response of $z_i^C$ are different, but become equivalent afterward, then $z_i^M$ is a curable output by the definition. On the other hand, if the value of $z_i^M$ before the injection and the response of $z_i^C$ are the same, but become different afterward, then $z_i^M$ is a new mismatched output. These numbers are calculated for computing *rank*1 and *rank*2 of signal $f$.

■ Step 4: *Restore the image* (i.e., the value of each signal) of the MODEL back to its fault-free one. During the execution of the loop in step 2, every new event including the initial event should be recorded in a data structure called *event history* to support an efficient roll back. For example, we only need to roll back the fault-free values of those signals that are touched during the event-driven fault simulation. Without such a support, one may need to roll back every signal's fault-free value. In that case, the time complexity of computing a signal's curable measures will become O($n$), where $n$ is the total number of signals in the MODEL, and the advantage of using event-driven simulation may disappear because the computation is now dominated by the roll-back process. After the roll back, the next fault-simulation run for another candidate signal can then be started.

At the end of this ranking heuristic, a condition is checked to decide if the program should have an early termination. This occurs when there exists a so-called *single fault*

*candidate*. A single fault candidate is a signal that can explain every output mismatch for every failing input vector. That is, every failing input vector is curable by this signal. When such a signal, say *f*, exists, then it can be shown that signal pair ($f$, $w_2$), where $w_2$ is any signal, will be a perfect double fault candidate 6. Then, the number of the perfect double fault candidates would be more than the total number of signals in the circuit under diagnosis. For such a case, finding one fault is still possible. The ranked list produced in Phase 1 serves as a guide. However, finding the second fault would be extremely difficulty due to the low resolution of the given test patterns. Therefore, our program will terminate early when this happens. To avoid such a situation, higher-quality test patterns (such as those diagnostic test patterns) might be used instead of random patterns.

**Phase 2: First-Level Filtering**

In the second phase, we apply a simple filtering heuristic based on an observation that the average first-hit index is usually very small, (e.g., an average of 3.0 for the entire set of ISCAS85 benchmark circuits). Therefore, it is very likely that the first few signals in the sorted signal list will contain at least one faulty signal. With this observation, we have the following selection criterion.

**Selection criterion:** Signal pair ($w_1$, $w_2$) is regarded as a *double fault candidate* if the index of $w_1$ in the sorted list after Phase 1 is smaller than or equal to a *selection threshold*. Signal $w_2$ could be any signal other than $w_1$.

By this criterion, the number of total fault candidates will be O(*selection-threshold* × *n*), where *n* is the total number of signals in the MODEL. With such a significant reduction in fault candidate space, we can afford to use more expensive techniques to check every candidate pair. The quality of this filtering heuristic depends on the selection threshold value. The larger this value, the higher success rate of having the real fault pair in the final candidate space. Since the average first-hit index still varies from one circuit to another. We use a *learning process* outlined in Fig. 5 to automatically set a good selection threshold.

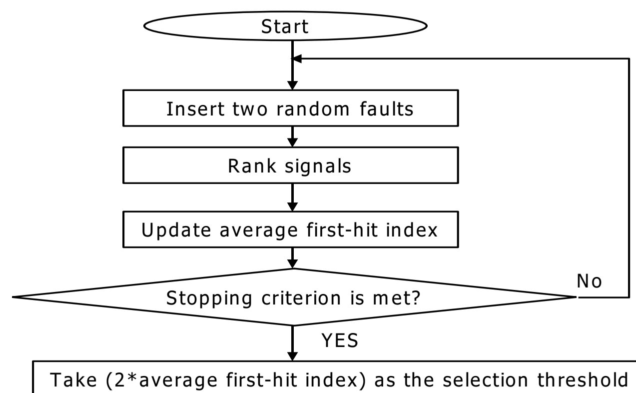


Fig. 5. A learning process to choose a good selection threshold.

Note that diagnosis is often performed after the first silicon is manufactured. While the learning process can be performed much earlier when the logic circuit is available. Hence, it does not necessarily add to the fault diagnosis time. The goal of this process is to characterize the good selection threshold of each block in the CUD. Once the fault is localized within a block through simple structural pruning, such as cone intersection [1], the selection threshold of the suspicious block can be utilized to retain a good balance between the diagnosis time and success rate. In computational detail, we generate a number of faulty circuits, each being created by injecting two random gate-type faults to the fault-free model. Whenever a new faulty circuit is created, the ranking heuristic in Phase 1 is applied to find its first-hit index. The average of the first-hit indexes of the faulty circuits that have been tried during this process is recorded. The termination condition of this learning process depends on when this average will converge. In our experiments, the stopping criterion is when the average first-hit index converges within a certain user-defined range.

**Phase 3: Second-Level Filtering**

In the third phase, we incorporate a more stringent and more computationally expensive filter. The objective is to further reduce the set of double fault candidates. We use the enumeration-based diagnostic technique in ErrorTracer [6] for this purpose. We use an example to explain this filtering technique. The readers are referred to the literature [6] for additional details.

**Example 4:** (Double fault candidate filtering) To decide if a signal pair ($w_1$, $w_2$) is a valid double fault candidate requires the search for a cure injection for every failing input vector. A cure injection is an injection at ($w_1$, $w_2$) such that every mismatched output is cured, while at the same time not creating any new mismatch. In the worst case, we have to try out four possible injections – {($w_1$, $w_2$) | (0, 0), (0, 1), (1, 0), (1, 1)}. For a candidate to be valid, there should be a cure injection at that signal pair for each failing input vector. Otherwise, it will be claimed as a false candidate and removed from the candidate list. In Fig. 6, injecting (0, 0) to a candidate signal pair ($w_1$, $w_2$) in the MODEL cures all mismatches. Therefore, the candidate ($w_1$, $w_2$) remains valid for the given input vector.

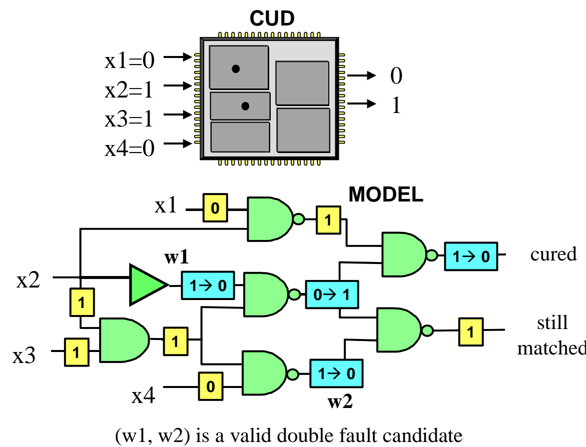


Fig. 6. Example of the double fault candidate filtering.

Unlike the filter in Phase 2, the enumeration-based diagnosis is a lossless filter in that it produces a small *potential set of candidates* (*P-set*) that will always contain the real faulty signal pair if it has passed the filtering in Phase 2. In other words, unlike the filtering in phase 2 which may cause the loss of accuracy, the filter at this phase will not overlook the real faulty signal pair. The output at this phase, i.e., the P-set, can be used directly to guide the physical failure analysis (e.g., silicon inspection), or, it can be further refined.

**Phase 4: Survivor Pair Ranking**

In the fourth phase, we use the ranking heuristic of phase 1 again. The signals being ranked are those signals contained in any signal pair of P-set. The number of such signals is usually small. After the ranking, we produce a *final sorted signal pair list* (*P-list*). This phase is motivated by the fact that the signal pairs in P-set have equal probability of being faulty – every one of them is able to cure every failing input vector. In order to guide the silicon inspection, a ranked list is more desirable. Hence, we define the *rank of a signal pair* ($w_1$, $w_2$) as the higher rank of signals $w_1$ and $w_2$. Based on this definition, we sort the survivor signal pairs to derive the final P-list.

**Example 5:** (Pair-ranking example) Suppose there are five survivor double fault candidates in the P-set, {($w_1$, *a*), ($w_1$, *b*), ($w_2$, *a*), ($w_2$, *c*), ($w_1$, $w_2$)}. Then, the suspicious faulty signals include {*a*, *b*, *c*, $w_1$, $w_2$}. Assume that the ranks of these signals produced in Phase 1 are {$w_1 = 1$, $w_2 = 2$, $c = 3$, $a = 5$, $b = 6$}. Then, the P-list generated by our program will be ($w_1$, $w_2$, *c*, *a*, *b*). The failure analysis can be conducted based on either the P-set or the P-list.

## 4. GENERIC BRIDGING FAULT MODEL

In this section, we proposed a *generic bridging fault model* that characterizes the physical faulty behavior more accurately than simple AND-bridging or OR-bridging fault models. One advantage of our diagnostic algorithm, described in section 3, is the ability to deal with such a generic bridging fault.

### 4.1 A Generic Bridging Fault Model

Fig. 7 shows the transistor schematic of a sub-circuit in which signal $\alpha$ and signal $\beta$ are accidentally connected. Suppose the bridging is not perfect and the resistance of the wire connecting the two signals is $R_{bridge}$. After bridging, the voltage levels of the two resulting signals are denoted as $V_{\alpha\text{-bridged}}$, and $V_{\beta\text{-bridged}}$. These two voltage levels depend on several factors. For example, they depend on the logical values of the input signals driving the gates at signal $\alpha$ and signal $\beta$. They are also dependent on the transistor sizes in the two logic gates and $R_{bridge}$. Precise computation may require simulation using SPICE. The effects of these two ambiguous signals could be interpreted as either logic '0' or logic '1' by their respective fanout gates. As a result, for a resistive bridging fault, the affected two signals could take on any one of the four possible combinations: ($\alpha = 0$, $\beta = 0$), ($\alpha = 0$, $\beta = 1$), ($\alpha = 1$, $\beta = 0$), and ($\alpha = 1$, $\beta = 1$). Taking all possibilities into account, a conservative model can be stated as follows.

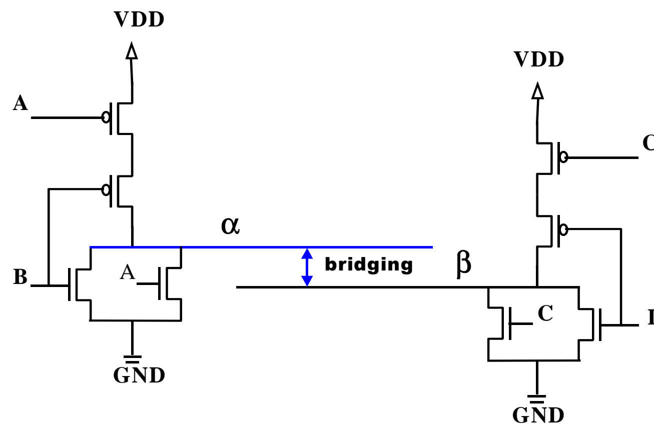


Fig. 7. A schematic with a bridging fault.

**Model:** (*Generic Bridging Fault Model*) Assume that a bridging fault occurs at signal α and β. If an input vector causes α and β to have different logic values in the fault-free circuit, then the new logic values of α and β due to the bridging could take on '0' or '1' independently in the faulty circuit. If an input vector causes α and β to be the same value, then both signals remain fault free.

In this model, there is a possibility that *both signals affected have fault effects*. Consider the example shown below.

**Example 6:** Let *v* be an input vector that causes α and β to be '0' and '1' in the fault-free circuit, respectively. Assume that signals α and β are bridged together and their resulting voltages, $V_{\alpha\text{-bridged}}$ and $V_{\beta\text{-bridged}}$, are both 2.5 volts. If the input threshold voltages of the immediate fanout gate of α and β are 2.0 and 3.0 volts, respectively, then, $V_{\alpha\text{-bridged}}$ could be interpreted as logical '1' (because $V_{\alpha\text{-bridged}} > 2.0$) while $V_{\beta\text{-bridged}}$ as logical '0' (because $V_{\beta\text{-bridged}} < 3.0$). In this case, both signals are *victims* of the bridging fault for the input vector *v*.

### 4.2 Generic Bridging Fault Injection Program

Based on the generic fault model, we design a *generic bridging fault injection* program to conduct our experiments. Note that this model is better than the simple AND-bridging or OR-bridging models. Furthermore, it can be enhanced to take the Byzantine fault phenomenon into account if the symbolic techniques we proposed in [8] are incorporated. The injection program first randomly selects two signals at which the bridging occurs. Note that we do not intend to deal with the feedback-type bridging fault 1 in this paper. Hence, if the two selected signals form any combinational cycles, we will make the selection again until the two selected signals do not form cycles. Then, we inject the bridging fault by constructing the new Boolean functions at the selected two signals. Just like the outline in Model 1, for input vectors that cause mismatches at α and β, we *randomly* assign '0' or '1' as the new values of the two signals' functions in the faulty circuit.

**Example 7:** (Illustration of a generic bridging fault) Suppose $w_1$ and $w_2$ are the two signals where a bridging occurs. In the fault-free circuit, signal $w_1$ is the output of a two-input NAND gate of signals A and B, while signal $w_2$ is the output of a two-input NOR gate of signals C and D. In the faulty circuit where $w_1$ and $w_2$ are shorted, the new functions of $w_1$ and $w_2$, denoted as $w_1^{new}$ and $w_2^{new}$, are functions of A, B, C, and D as shown in Fig. 8. For input combinations of signals A, B, C, and D that activate the bridging fault, $w_1^{new}$ and $w_2^{new}$ are independently assigned a random binary value. Otherwise, their values remain fault free. In other words, the function $w_1^{new}$ and $w_2^{new}$ of Fig. 8 is generated by the following C-code segment:

```
if(w1(A,B)==w2(C,D)) {
      w1new(A,B,C,D) = w1(A,B);
      w2new(A,B,C,D) = w2(C,D);
}
else {
      w1new(A,B,C,D) = random()%2;
      w2new(A,B,C,D) = random()%2;
}
```

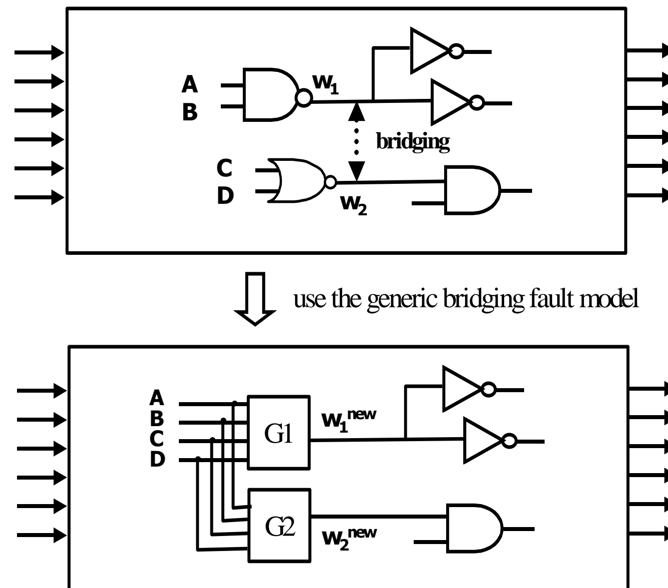


Fig. 8. Illustration of a generic bridging fault model.

## 5. EXPERIMENTAL RESULTS

The proposed algorithm has been implemented using the C language in the SIS environment 4. The experiments are performed on Ultra-SPARC workstation running at 450 MHz. For every ISCAS85 benchmark circuit, we generate 100 faulty circuits of each of the following three fault types through random fault injection programs.

- *Two stuck-at faults*: two signals are selected and set to either the '0' or '1' stuck-at values.

- *Two random gate-type faults*: Two gates are randomly selected whose Boolean functions are scrambled randomly by a program.

- *A generic bridging fault* using the program mentioned in section 4.2.

Table 1 shows the first-hit indexes for the diagnosis of different types of faulty circuits. On average, the indexes are 3.3 for circuits with two stuck-at faults, 3.0 for circuits with two gate-type faults, and 3.1 for circuits with one generic bridging fault. This indicates that our algorithm is indeed not restricted to any fault model.

**Table 1. First-hit of different types of faults.**

| circuit | # nodes | 2 stuck-at | 2 gate-type | 1 bridging |
|---|---|---|---|---|
| C432 | 175 | 2.4 | 1.7 | 2 |
| C499 | 370 | 52. | 3.7 | 3.3 |
| C880 | 302 | 3 | 2.9 | 2.6 |
| C1355 | 474 | 4.3 | 4.6 | 4.7 |
| C1908 | 465 | 3.9 | 3.4 | 4.4 |
| C2670 | 694 | 3.5 | 3.9 | 3.6 |
| C3540 | 1020 | 2.6 | 2.7 | 2.9 |
| C5315 | 1462 | 2.3 | 2.5 | 2.23 |
| C6288 | 2353 | 1.9 | 1.7 | 1.3 |
| C7552 | 2114 | 3.8 | 3.2 | 3.4 |
| **average** | **942.9** | **3.3** | **3.0** | **3.1** |

Table 2 compares the accuracy of the ranking heuristic used in Phase 1 to the results presented in 7. On average, the first-hit index has been improved from 5.3 to 3, or a 43% improvement. Note that if only one fault is to be searched, then this index means that on average one can find one faulty signal after examining the silicon for 3 signals during the failure analysis. Note that we only compare the results to our previous work is because there are no other published results to our knowledge in this newly emerging area that address the same problem using the same measure.

Table 3 shows the diagnostic results of faulty circuits with two gate-type faults. The second-hit index, the success rate, and the CPU times are shown, where the second-hit index means the index of the second real faulty signal in the final sorted list of signals. Compared to 6, the CPU time has been improved by at least *one order of magnitude*, at the cost of missing only 3.7% of the cases diagnosed (i.e., the success rate is 96.3%). With such a heuristic, the simultaneous diagnosis of double faults is then feasible

within reasonable CPU time. Furthermore, in this framework, one can explore the trade-off between the CPU time and the success rate of finding the second fault."

**Table 2. Results of ranking heuristic in Phase 1.**

| circuit | # nodes | 1 st-hit | 2 gate-type | time (sec) |
|---|---|---|---|---|
| C432 | 175 | 2.8 | 1.7 | 1 |
| C499 | 370 | 8.9 | 3.7 | 3 |
| C880 | 302 | 3.4 | 2.9 | 2 |
| C1355 | 474 | 14.1 | 4.6 | 4 |
| C1908 | 465 | 4.8 | 3.4 | 5 |
| C2670 | 694 | 5.4 | 3.9 | 16 |
| C3540 | 1020 | 2.4 | 2.7 | 9 |
| C5315 | 1462 | 3.2 | 2.5 | 32 |
| C6288 | 2353 | 1.9 | 1.7 | 51 |
| C7552 | 2114 | 6.3 | 3.2 | 46 |
| **average** | **942.9** | **5.3** | **3.0** | **16.9** |

**Table 3. Results of diagnosing circuits with two gate-type faults.**

| circuit | 2nd-hit | success rate | time [6] | our time |
|---|---|---|---|---|
| C432 | 6.1 | 98% | 5 | 2 |
| C499 | 9.4 | 91% | 13 | 5 |
| C880 | 8.5 | 98% | 8 | 4 |
| C1355 | 14.9 | 92% | 7 | 11 |
| C1908 | 13.2 | 97% | 16 | 11 |
| C2670 | 12.5 | 97% | 21 | 18 |
| C3540 | 8.6 | 97% | 137 | 26 |
| C5315 | 11.8 | 97% | 141 | 49 |
| C6288 | 5.4 | 97% | 3244 | 120 |
| C7552 | 18 | 99% | 1168 | 142 |
| **average** | **10.8** | **96.3%** | **476.0** | **38.8** |

## 6. CONCLUSIONS

In this paper, we investigated a new approach for double fault diagnosis of combinational circuits. The goal is to catch both faults simultaneously. To address the problem of an explosion in the number of candidates (i.e., the candidate number is proportional to the square of the size of the circuit under diagnosis), we proposed an improved ranking based heuristic to quickly discard unlikely candidate signal pairs. In this way, we trade accuracy for efficiency. The diagnosis time for every ISCAS85 benchmark circuit can be done in a few minutes. Experimental results also show that the real fault pair is rarely overlooked by such a heuristic. On average, the success rate is as high as 96.3%, where the success rate is the chance of including the real fault pair in the final list of suspect signal pairs produced by our algorithm at the end of diagnosis. We tested the algorithm on circuits with two independent gate-type faults, on circuits with two stuck-at faults, and on circuits with a bridging fault. The results are all quite similar. Therefore, we conclude that the proposed approach is not only efficient and accurate, but also applicable to a wide variety of faults.

## REFERENCES

1. M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital System Testing and Testable Design*, Computer Science Press, 1990.
2. M. Abramovici and M. A. Breuer, "Multiple fault diagnosis in combinational circuits based on an effect-cause analysis," *IEEE Transactions on Computers*, Vol. C-29, 1980, pp. 451-460.
3. R. C. Aitken, "Modelling the unmodellable: Algorithmic fault diagnosis," in *Proceedings of International Test Conference*, 1996, pp. 931-940.
4. R. K. Brayton et. al, "SIS: A system for sequential circuit synthesis," Technical report, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, 1992.
5. F. Celeiro, L. Dias, J. Ferreira, M. B. Santos, and J. P. Teixeira, "Fault-oriented IC test and diagnosis using VHDL fault simulation," in *Proceedings of International Test Conference*, 1996, pp. 620-628.
6. S.-Y. Huang, K.-T. Cheng, K.-C. Chen, and D.-T. Cheng, "ErrorTracer: A fault simulation based approach to design error diagnosis," in *Proceedings of International Test Conference*, 1997, pp. 974-981.
7. S.-Y. Huang, "On improving the accuracy of multiple fault diagnosis," in *Proceedings of VLSI Test Symposium*, 2001, pp. 34-39.
8. S.-Y. Huang, "Speeding up the Byzantine fault diagnosis using symbolic simulation," in *Proceedings of VLSI Test Symposium,* 2002, pp. 193-198.
9. J. B. Khare, W. Maly, S. Griep, and D. Schmitt-Landsiedel, "Yield-oriented computer-aided fault diagnosis," *IEEE Transactions on Semiconductor Manufacturing*, Vol. 8-2, 1995, pp. 195-206.
10. I. Pomeranz and S. M. Reddy, "On correction of multiple design errors," *IEEE Transactions on Computer-Aided Design*, Vol. 14, 1995, pp. 255-264.
11. P. G. Ryan, "Logical diagnosis solutions must drive yield improvement," in *Proceedings of International Test Conference*, 1997, pp. 434-442.
12. S. Narayanan, R. Srinivasan, R. P. Kunda, M. E. Levitt, and S. Bozorgui-Nesbat, "A fault diagnosis methodology for the ultraSPARC/sup TM/-I microprocessor," in *Proceedings of European Design and Test Conference*, 1997, pp. 494-500.
13. S. Venkataraman and S. B. Drummonds, "A technique for logic fault diagnosis of interconnect open faults," in *Proceedigns of VLSI Test Symposium*, 2000, pp. 313-318.
14. S. Venkataraman and S. B. Drummonds, "POIROT: A logic fault diagnosis tool and its applications," in *Proceedings of International Test Conference*, 2000, pp. 253-262.

**Hong-Chou Kao (高宏州)** received his B.S. degree from National Central University, Taiwan in 1999, and M.S. degree from National Tsing-Hua University, Taiwan, in 2001. Both degrees are in electrical engineering. Soon after his graduation, he joined Industrial Technology Research Institute as an engineer in Computer and Communication Lab. His technical expertise includes digital VLSI design, testing, and diagnosis.

**Ming-Fu Tsai (蔡名甫)** received his B.S. and M.S. degrees from National Tsing Hua University, Taiwan, in 1999 and 2001, respectively. Both degrees are in electrical engineering. In 2001, he joined Sota Design Technology Inc.. His technical expertise includes VLSI design and testing.

**Shi-Yu Huang (黃錫瑜)** received his BS, MS degrees in Electrical Engineering from National Taiwan University in 1988, 1992 and Ph.D. degree in Electrical and Computer Engineering from the University of California, Santa Barbara in 1997, respectively. He joined the faculty of National Tsing-Hua University, Taiwan ROC, in 1999, where he is currently an Associate Professor. His research interests are in the area of computed-aided design for VLSI, with an emphasis on design verification, diagnosis, and testing. He co-authored a book entitled "Formal Equivalence Checking and Design Debugging" published by Kluwer Academic Publishers in 1998.

**Cheng-Wen Wu (吳誠文)** received the BSEE degree in 1981 from National Taiwan University, Taipei, Taiwan, and the MS and PhD degrees in electrical and computer engineering, in 1985 and 1987, respectively, from the University of California, Santa Barbara. Since 1988 he has been with the Department of Electrical Engineering, National Tsing-Hua University (NTHU), Hsinchu, Taiwan, where he is currently a professor. He also has served as the director of the university's Computer and Communications Center from 1996 to 1998, and the director of the university's Technology Service Center from 1998 to 1999. From August 1999 to February 2000, he was a visiting faculty of the ECE Department, UCSB. Since August 2000, he has been the Chair of the EE Department of NTHU. He also is the Director of the IC Design Technology Center of the university.

Dr. Wu was the Technical Program Chair of the IEEE Fifth Asian Test Symposium (ATS'96), and the General Chair of ATS'00. He is an Associate Editor for the Journal of the Chinese Institute of Electrical Engineers (JCIEE) and a Guest Editor of the JCIEE Special Issue on Design and Test of System-on-Chip, and was a Guest Editor of the Journal of Information Science and Engineering (JISE), Special Issue on VLSI Testing. He received the Distinguished Teaching Award from NTHU in 1996, the Outstanding Electrical Engineering Professor Award from the Chinese Institute of Electrical Engineers (CIEE) in 1997, the Distinguished Research Award from National Science Council

in 2001, the Industrial Collaboration Award from the Ministry of Education in 2001, and the Best Paper Award from the 2002 IEEE International Workshop on Design & Diagnostics of Electronic Circuits & Systems. He is interested in design and testing of high performance VLSI circuits and systems. Dr. Wu is a life member of CIEE and a senior member of IEEE.

**Wen-Feng Chang (張文峰)** received his Ph.D. degree from National Tsing-Hua University, Taiwan, in 2000. He was with Syntest Technology Inc. in 2001, working on IC diagnostic tools. His specialty is mainly in the areas of VLSI testing and VLSI design automation.

**Shyue-Kung Lu (呂學坤)** received his M.S. degree from National Tsing-Hua University, Taiwan, in 1991, and Ph.D. degree from National Taiwan University, Taiwan, in 1996. Since 1998, he has been an Associate Professor in Fu-Jen Catholic University, Taiwan. He is an active member of VLSI Test Technology Forum in Taiwan. His research interests include VLSI testing, fault tolerance, VLSI design, and computer system.