FAILURE ANALYSIS FOR FULL-SCAN CIRCUITS

Kaushik De and Arun Gunda
LSI Logic Corporation
1501 McCarthy Blvd., M/S E-192
Milpitas, CA 95035, USA
{kaushik, arun}@lsil.com

Abstract

We present a complete system for failure analysis of full-scan circuits. A novel scheme has been proposed to handle multiple faults up to a certain extent by ranking the faults according to the likelihood of being present in the defective part. The user can interactively recompute the suspect fault list by changing some parameters. If the suspect fault list is large, we generate new test patterns to distinguish the faults in the suspect list. User can iterate over a defective part several times until the suspect fault list is reasonably small. Then each suspect site is probed using E-beam. This tool is integrated into design environment of LSI Logic Corporation and produced good results when applied on a few industry circuits.

1 Introduction

Failure analysis is an important step in improving profitability by identifying and rectifying the recurring defects in the integrated circuits. As the device sizes shrink and new package types like flip-chip are introduced, automated tools for failure analysis are gaining importance.

Researchers have studied the failure analysis problem from the sixties and early seventies [1]. Most of the work in diagnosis has been based on classical single stuck-at fault model, and has been limited to the combinational circuits [2]. The concept of *fault dictionary* has been introduced, which is a record of the errors a circuits modeled faults are expected to cause [3]. The size of fault dictionary can be very large. Techniques have been developed to compress the size of fault dictionary [4, 5]. New test generation technique has been developed to generate test vectors to distinguish a pair of faults [6, 7, 8]. Since many likely physical defects cannot be accurately modeled by the stuck fault model, recent work has addressed the problems with the accuracy of diagnosis of bridging faults when the fault dictionary was based on stuck-at fault model [9]. Also, research has been performed in the area of extension of fault dictionary to the sequential circuits [10]. Matching Algorithms have been developed to include potential detections in sequential circuits and to rank those matches [11]. Fault diagnosis using static current monitoring in CMOS circuits is developed in [12, 13], which shows substantial improvement in diagnostic resolution.

Fault diagnosis for a full-scan design has been reported in [2]. No fault dictionary is created beforehand. From the failure information from the tester, structural and parity analysis is performed and a subset of the total faults which are consistent with the analysis are chosen for fault simulation. There are three different classes of failures user can choose when performing fault localization, *single stuck fault*, *non-SSF/single-site defect* and *non-SSF/multiple-site defect.* Fault location for full-scan circuits using partial intersection method with threshold is presented in [14]. The problem of correcting errors in macro-based circuits has been considered in [15]. Both design errors and physical circuit errors (due to manufacturing defect or error in customizing a programmable circuit) are considered in this work and both single and multiple errors are considered.

In this paper, we present a complete system for failure analysis of full-scan circuits. We use the stuck-at fault model in our analysis. We present a novel scheme of fault localization which can handle multiple faults up to a certain extent. We rank the faults according to their *merit values*, which measures the likelihood of that fault being present. We also present a scheme of generating new test patterns to distinguish the faults in the suspect

0-7803-2991-0/95 $4.00 © 1995 IEEE

fault list if the suspect fault list is large. Preliminary report of this system, highlighting the electron beam probing procedure, is described in [16, 17].

2 Fault Localization

2.1 Definitions

Let us denote the set of test vectors be V, comprises of m test patterns,

$$V = \{v_1, v_2, \ldots, v_m\}.$$

Let us denote the set of stuck-at faults of the design under test (DUT) *detected* by V be F, comprises of n faults,

$$F = \{f_1, f_2, \ldots, f_n\}.$$

Let us denote $\Delta(v)$ be the set of faults detected by the vector v and $\Theta(f)$ be the set of vectors which detect the fault f.

For example, if the vector v_i detects the faults f_1, f_7 and f_{15}, then $\Delta(v_i) = \{f_1, f_7, f_{15}\}$. As a result, v_i will be a member of $\Theta(f_1)$, $\Theta(f_7)$ and $\Theta(f_{15})$

The set of failing patterns, the test patterns for which the part behaves erroneously, is denoted as FP.

The ***phase*** of a path is said to be ***negative*** (***positive***) if the number of signal inversions along the path is an odd (even) number.

The site of a fault f is said to be in ***positive input cone*** (***negative input cone***) of an output *out*, $PIC(out)$ ($NIC(out)$), if there exist a path of ***positive phase*** (***negative phase***) from the fault site to *out*.

For every erroneous output o for a faulty pattern v, we define a set of ***consistent phase fault***, $CPF(o, v)$, as follows. A fault f is a member of $CPF(o, v)$ if one of the following two conditions is true:

1. faulty value of f is the same as the erroneous value of o and f is a member of $PIC(o)$.

2. faulty value of f is the opposite of the erroneous value of o and f is a member of $NIC(o)$.

For example, a fault type stuck-at-0 will be a member of consistent phase fault for an erroneous output O_1 and vector v if either the erroneous value of O_1 is 0 and there is a positive phase path from the fault site to O_1, or erroneous value of O_1 is 1 and there is a negative phase path from the fault site to O_1.

When a faulty part of the design is tested with the test set V, let $\Phi(v)$ be the set of primary and scan outputs where error is observed when a test pattern $v \in FP$ is applied.

Let us denote NFO be the total number of erroneous outputs summed over all the failing test patterns,

$$NFO = \sum_{v \in FP} ||\Phi(v)||,$$

where $||S||$ denotes the cardinality of the set S.

For a test pattern v, the number of output errors which are potentially due to the fault f, *pot_cause*(f, v), is defined as

$$pot\_cause(f, v) = ||\{o \in \Phi(v) \mid f \in CPF(o, v)\}|| \quad (1)$$

It measures the number of erroneous outputs for the vector v which are reachable from the site of the fault f and the erroneous values are consistent with the faulty value of the fault. In other words, it is the number of erroneous outputs which are potentially caused by the presence of the fault f. For example, in Figure 1, the erroneous outputs and the erroneous values are shown for a vector v. There are 6 erroneous outputs which are reachable from the fault site of f. However, in order to cause an erroneous value of 1 at o_3 by the fault f (stuck-at-0), there must exist a negative phase path from the site of f to o_3. We can observe in the figure that such a path exists, hence f is a member of $CPF(o_3, v)$. On the other hand, in order to produce an erroneous value of 0 at o_{10}, there must exist a path of positive phase from the site of f to o_{10}. But there is only one path from f to o_{10}, and that is a negative phase path. Hence, the fault f ***cannot*** be the cause of erroneous value of 0 at the output o_{10}. The erroneous outputs which are consistent with the fault f are o_3, o_5, o_{17} and o_{19}. These are the erroneous outputs which are potentially caused by the fault f. Hence, *pot_cause*$(f, v) = 4$.

2.2 Merit Scale of the Faults

When a part is tested in a tester and found to be faulty, the tester provides the following information.

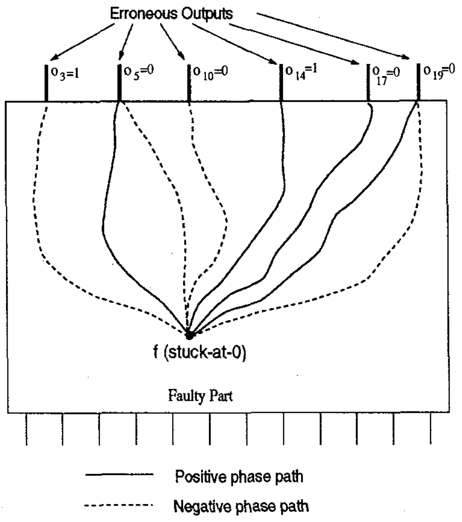

Figure 1: An example of calculation of pot_cause for a fault f and a vector v

1. The list of failing vectors
2. The list of erroneous outputs (both primary outputs and scan outputs) for each failing vector.

From the above information, we want to rank the faults in the decreasing likelihood of presence in the faulty part. If a fault is ranked higher, its expected behavior matches more closely with the actual behavior of the faulty circuit. The behavior of a faulty circuit can be categorized into two groups:

1. The test patterns in which some of the outputs (at least one) have erroneous values - we denote this set of test patterns as *erroneous patterns* and this part of the behavior as *erroneous behavior*.
2. The test patterns in which no erroneous output is present - we denote this set of test patterns as *error-free patterns* and this part of behavior as *error-free behavior*.

If the expected behavior of a fault f matches exactly with both the *erroneous behavior* and the *error-free behavior* of the faulty circuit, then the presence of fault f *alone* can explain the complete behavior of the faulty circuit. Such a fault is denoted as a *suspect fault under single fault assumption*.

If the fault can explain only a part of the *erroneous behavior* partly, in order for the fault to be present in the faulty part, some other fault(s) *must* also be present in the faulty part. Similarly, if the *error-free behavior* does not match exactly with the behavior of a fault, it must have been *masked* by another fault present in the circuit if that fault is present. Those faults are denoted as *suspect faults under multiple fault assumption.*

We need to measure quantitatively how close the expected behavior of each fault matches the behavior of the faulty part. We compute two values for each fault in the circuit. The first value, *Detect(f)*, computes the number of failing outputs which are reachable from the fault site of the fault f with consistent phase for all the failing patterns. It is defined as

$$Detect(f) = \sum_{v \in FP} pot\_cause(f,v) \qquad (2)$$

This value measures how closely the behavior of the fault f matches with the *erroneous behavior* of faulty part. If the value *Detect(f)* is equal to NFO, then *all* the erroneous outputs are reachable from the fault site of f with consistent phase, and can potentially be caused by the presence of the fault f. If *Detect(f)* is not equal to NFO, then (NFO - *Detect(f)*) erroneous outputs can not be caused by the presence of the fault f.

The second value, *NoDetect(f)*, measures how closely the expected behavior of a fault f matches with the *error-free* behavior of the faulty part. It measures the number of times the effect of the fault f *should have been* observed if the fault was present, but *was not* observed. This behavior can be categorized into two groups:

1. The set of test patterns which detect the fault f, but the faulty part did not behave erroneously for those patterns i.e., fault effect of f was not observed.
2. The set of patterns which detect the fault f, but all the erroneous outputs were *not* reachable from the fault site, i.e., the fault effect of f was not observed.

Hence, *NoDetect(f)* is defined as

$$\begin{aligned} NoDetect(f) &= \|\Theta(f) \bigcap (V - FP)\| + \\ &\|\Theta(f) \bigcap \{v \in FP \,|\, pot\_cause(f,v) = 0\}\| \end{aligned} \quad (3)$$

If *NoDetect(f)* is equal to 0, then the behavior of f matches exactly with the *error-free* behavior of the faulty part. If it is non-zero, then in order to the fault be present in the faulty part, it *must* be masked by other faults in *NoDetect(f)* number of test patterns.

The merit of a fault f, *Merit(f)*, measures how closely the expected behavior of a fault matches with the complete behavior of the faulty part. It is defined as

$$\begin{aligned} Merit(f) &= C_1 \star (NFO - Detect(f)) + \\ &\quad C_2 \star NoDetect(f) \end{aligned} \quad (4)$$

where C_1 and C_2 are user defined constants and signifies relative importance of matching the *erroneous behavior* and the *error-free behavior* of the faulty part, respectively.

Theorem 1 *Under single stuck-at fault assumption, a fault f can be present in the defective part only if the fault f has Merit value 0 in Equation (4) for any non-zero positive values of C_1 and C_2.*

Proof: If a fault f has non-zero Merit value, then either Detect(f) is not equal to NFO or NoDetect(f) is non-zero. If Detect(f) $\neq NFO$, then there exist an erroneous output which can not be caused by f. Hence some other fault must be present in the circuit, which violates the single stuck-at fault assumption. If NoDetect(f) is non-zero, the fault effect of f was not observed on atleast one vector which detects f. Which can only happen if f is masked by some other fault in the circuit, which again violates the single fault assumption. ■

The theorem above states that if a fault f has non-zero Merit value for any non-zero positive values of C_1 and C_2, then that fault *cannot* be present in the circuit under single fault assumption. However, if any fault has zero Merit value for any non-zero positive values of C_1 and C_2, it may not be physically possible for the fault to be present in the circuit. This is because our procedure of computing the phase of a path is approximate, as it will be clear in the next subsection. However, our procedure of computing *Detect*(f) and *NoDetect*(f) is such that it is the upper bound of the actual *Detect* value of that fault and the lower bound of the actual *NoDetect* value of that fault. Hence, the above theorem holds true even for approximate values of *Detect* and *NoDetect*.

The Merit scale is designed such that it ranks all the possible stuck-at faults in the design in decreasing likelihood of being present in the defective part. The lower the value of *Merit(f)*, the higher the chance that the fault may be present in the faulty part.

Usually, the chance of masking of fault effect of a fault is very low. Hence, C_2 should be assigned to a large number. However, the maximum value of (*NFO - Detect(f)*) is NFO. Hence, there is no need for assigning the value of C_2 to be any higher than NFO if the value of C_1 is assigned to 1. The possible scenarios are explained in Figure 2 for $C_1 = 1$ and $C_2 = NFO$. For those values, the faults which have merit values in the range $[k \star NFO, (k+1) \star NFO]$ can be present in the circuit if and only if there exist multiple faults in the circuit and each of those faults are masked in exactly k vectors.

Theorem 2 *If we assume that there are at most K stuck-at faults present in the circuit with no masking*

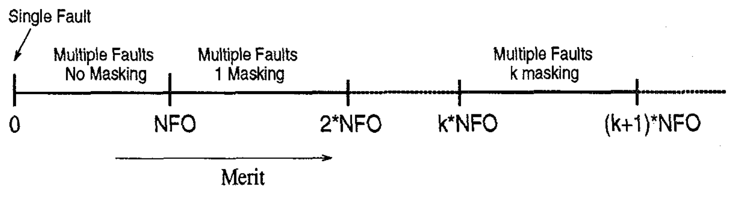
Single Fault
Multiple Faults No Masking
Multiple Faults 1 Masking
Multiple Faults k masking
0
NFO
2*NFO
k*NFO
(k+1)*NFO
Merit

Figure 2: Different scenarios on merit scale for $C_1 = 1$ and $C_2 = NFO$

of fault effects, then the set of faults with Merit value less than equal to $\lfloor \frac{(K-1)*NFO}{K} \rfloor$ *with* $C_1 = 1$ *and* $C_2 = NFO$ *contains at least one of the faults present in the circuit.*

Proof: There are NFO erroneous outputs in total, over all the failing patterns. Each of these erroneous outputs must be caused by one or more stuck-at-faults present in the circuit. Since there are at most K faults present in the circuit, by using pigeon-hole principle, there exists one fault f present in the circuit, which must explain at least $\lceil \frac{NFO}{K} \rceil$ erroneous outputs. As a result Detect(f) $>= \lceil \frac{NFO}{K} \rceil$. Since there is no masking, NoDetect(f) must be 0. Hence, by using Equation (4), Merit(f) = (NFO - Detect(f)) $<= \lfloor \frac{(K-1)*NFO}{K} \rfloor$. Hence, such a fault will be included in the set mentioned. ■

2.3 Fault Localization Algorithm

When a defective part is tested on a tester, it produces a list of test patterns for which the part has failed. It also lists the primary outputs and the flip-flops where errors were observed.

In the very first run on a design, the fault dictionary is created by performing fault simulation of all the faults for all the test vectors. The fault dictionary is stored in a file, and in all the later runs on the same design (different defective parts), the fault dictionary is read back from the file.

First, the *Detect* and *NoDetect* values are reset to zero for all the faults in the circuit. The algorithm then reads the failure information from the tester. It reads failing vectors and the erroneous outputs corresponding to each failing vector. It maintains two count value at each gate of the circuit, *positive count* and *negative count*. For every failing vector, it first resets the count values at each gate to 0. Then from each erroneous outputs, it traces back to the inputs. It starts with positive phase from the erroneous output if the erroneous value is 1, otherwise it starts with negative phase. If a gate is visited with positive (negative) phase, then it means stuck-at-1 (stuck-at-0) faults at the output of that gate can potentially explain the erroneous value at the erroneous output.

While tracing back from the output of a gate to an input of that gate, it keeps track if any signal inversion can happen through that path. For example, there is a signal inversion when tracing back from the output to the inputs of a NAND or NOR gate, where as there is no signal inversion for an AND or OR gate. If signal inversion happens, then the phase at the input of the gate is the opposite of the phase at the output of the gate, otherwise the phase at the input is the same as the phase at the output.

For some gates, the signal inversion can happen at some inputs depending on the values at the other inputs. For example, for a two input EX-OR gate, signal inversion happens from the output to an input if the value at the other input is 1, no signal inversion happens if the value at the other input is 0. For this kind of gates, it marks the input as both positive and negative phase and every gate in its input cone.

If a gate is visited with positive phase, then its *positive count* is incremented by 1, similarly for the negative phase. If it is visited with both the phases, both counts are incremented.

The algorithm then goes through all the faults which are detected by this failing vector v in the *fault dictionary*. For each fault f which is detected by the vector v, if the fault type is stuck-at-1 (stuck-at-0), then the *positive count* (*negative count*) of the driver gate of the fault site is added to *Detect(f)*. This is due to the fact that if the fault type is stuck-at-1 (stuck-at-0), then *pot_cause*(f, v) is equal to *positive count* (*negative count*) of the driving gate, because *positive count* (*negative count*) of a gate is the number of erroneous outputs for a failing vector v that can be potentially explained by a stuck-at-1 (stuck-at-0) fault at the output of that gate. If a stuck-at-1 (stuck-at-0) fault f is detected by the vector v but the *positive count* (*negative count*) of the driver of the node is 0, then the *NoDetect(f)* is incremented by 1. This is because the fault f can not be the cause of any of the erroneous outputs and it must be masked by some other fault if f is present in the faulty circuit.

Next, the algorithm analyzes the passing vectors. If faulty circuit fails in vector v_i and the next failure happens at the vector v_j, then the vectors v_{i+1}, v_{i+2}, ..., v_{j-1} have passed. If the first failing vector is v_f, then the vectors v_1, v_2, ..., v_{f-1} have passed.

For each passing vector v, the algorithm goes through all the faults detected by v and increments the *NoDetect* values of those faults by 1. This is because if a fault f is detected by a passing vector v, then f can be present in the faulty circuit only if it is masked by another fault in the circuit in the vector v.

Since we use only structural analysis, ignoring the actual logic values at different nodes, we will never miss any *real* path for any test vector, but we may mark some *false* paths. It is clear from the above description of computation of phase of a path that if a gate is visited from two paths with different polarities because of reconvergent fanout, then that gate is marked both *positive phase* and *negative phase*. As a result, the *Detect* value computed for any fault is the upper bound for the actual *Detect* value for that fault. Similarly, the computed *NoDetect* value is the lower bound of the actual *NoDetect* value. Hence, The computed *Merit* value of a fault is the lower bound of the actual *Merit* value of the fault for any non-zero positive values of C_1 and C_2.

After the analysis of all the passing and the failing vectors, the merit values of all the faults are calculated by using Equation 4, where C_1 and C_2 are user supplied parameters. Then the algorithm computes the suspect faults list as follows,

$$\mathit{susp\_flt\_list}(\mathit{threshold}) = \{f \in F | \mathit{Merit}(f) <= \mathit{threshold}\}$$

where *threshold* is a user supplied parameter. The set *susp_flt_list(0)* contains a set of faults each of whose presence *alone* can explain the complete behavior of the faulty circuit. If the user is not satisfied with that set, he/she can change the value of *threshold*, C_1 and C_2 to recompute the *susp_flt_list*. The algorithm for fault localization is shown in Figure 3.

2.4 Fault in Scan Chain

The fault localization algorithm described in this section can be applied if the scan chain itself does not have any fault in it. But if there is any fault in the scan chain, wrong inputs will be applied to the circuit, and the algorithm will not be able to diagnose any fault.

Hence, we devised a method of testing the scan chain and the data shifting mechanism of the scan chain before any scan pattern is applied. We shift a vector through the scan chain with alternating 0s and 1s in the vector. This vector is just shifted through the scan chain, no system clock is applied, and the data coming out of the scan chain is compared with the input data. This vector detects all stuck-at faults at the inputs and the outputs of the flip-flops.

If the scan check pattern fails, then the algorithm recognizes that there is a fault in the scan chain. It informs the user regarding that and quits. It does not analyze other failing vectors. The user then have to probe the scan chain with electron beam prober.

The scheme of probing scan chain is shown in Figure 4 with a scan chain of length 8. This scheme is identical to the binary search scheme. The scan check pattern is continuously shifted through the scan chain and electron beam probing is used. Initially, the middle of the scan chain is probed, which is the output of the cell 4. If the net is found error-free, the error must be on the right hand side of *probe1*, between the cells 5 and 8. The middle of that segment, the output of cell 6 is probed next. If the signal at that net is found faulty, the fault is in the segment between *probe1* and *probe2*. The middle of that segment, the output of cell 5 is probed. If it is found faulty, the fault is in the cell 5, otherwise the fault is in cell 6. After that, the internal of the particular cell needs to be probed to identify the actual defect. In general, the number of probing needed to identify the faulty cell is $\lceil \log_2 n \rceil$, where n is the length of the scan chain.

3 Iteration Over a Suspect Fault List

3.1 Distinguishing Pattern Generation

In the last section, we described the fault localization methodology to generate a suspect fault list. However, the suspect fault list can be very large. In that case, it will be very tedious to probe each fault site in the suspect list to locate the actual defect location. Also, this procedure is very expensive because it requires the use of an expensive tester and an electron beam prober for a long time.

Hence, if the suspect fault size if large, the user can choose to generate some more new test vectors, specifically targeted to distinguish the faults in the suspect list. These new test vectors are used to test the defective part, and using the *tester datalog* (tester output showing the list of failing vectors and corresponding erroneous outputs), we can further narrow down the suspect fault list.

The algorithm for distinguishing pattern generation attempts to generate test vector in such a fashion that each test vector is generated to detect a target fault in the suspect fault list and as few as possible other faults in the suspect fault list. Whenever a test vector is generated, all the faults are simulated. The pattern generation terminates when the maximum size of indistinguishable faults reaches a user defined threshold, or a time limit suppllied by the user expires.

3.2 Iteration

The newly generated patterns are applied on the defective part and the tester datalog is collected. In the next iteration on the defective part, only those faults are considered which were in the suspect fault list in the previous iteration. The tester datalog is analyzed and new and smaller suspect fault list is created by using the technique presented in Section 2. If the fault localization is not sufficient, new distinguishable patterns are generated and another iteration is performed. This process continues until the user is satisfied that the fault localization is small enough so that each individual fault site in the suspect fault list can be probed by using an electron beam prober to identify the actual defect location.

4 Diagnostic Pattern for Electron Beam Probing

Once the suspect fault list is small enough, all the location in the suspect fault list need to be probed in order to identify the actual defect location. Usually, the probing is done using an electron beam prober. Electron beam probing has been used for fault detection in IC as reported in [18, 19, 20, 21].

When the user chooses a net from the list of suspect fault locations, the tool generates a set of two patterns which assigns opposite logic values on that net. using these two patterns, the user can probe that net using an E-beam prober to test if that net is switching or not. If that net does not switch logic values when those two patterns are applied, that net has a stuck-at fault. Otherwise, the user chooses another net from the suspect fault list to repeat this procedure. This process continues until the user has identified the actual defect location.

5 Results

A complete system is developed as a part of the C-MDE^{TM} design environment of LSI Logic Corporation by implementing the algorithms described in the previous sections, and is named SCAN-FA tool. This tool is developed to perform failure analysis on full-scan circuits. However, it can also handle "almost full-scan" circuits as well.

To test the effectiveness of the algorithm, we applied the tool on two test cases. We will describe the results in this section. We are currently performing more experiments with this system on more test cases.

5.1 Test Case 1

A design for Company A is chosen as the first test case for this tool. This design has roughly 60,000 gates. The design was implemented on 100K gate array technology of LSI Logic Corporation with 2 layers of metal. It has some memory modules, which are black-boxed for the purpose of test generation. The length of the scan

```
Algorithm: Fault Localization ($threshold$, $C_1$, $C_2$)
/* Initialization */
For all faults $f \in F$
    Detect($f$) = NoDetect($f$) = 0;
last_fail_vec = 0;
NFO = 0;

/* Process failing vectors */
For all failing vector $v \in FP$
    /* Process Passing vectors */
    For passing vectors $p$ = last_fail_vec+1, ..., $v$-1
        For faults $f \in \Delta(p)$
            NoDetect($f$) += 1;

    For all gates $g$ in circuit
        positive_count($g$) = negative_count($g$) = 0;
    For all erroneous outputs $o \in \Phi(v)$
        NFO++;
        if(erroneous_value($o$) == 1) phase = POSITIVE;
        else phase = NEGATIVE;
        Trace_and_mark_input_cone($o$, phase)
    For faults $f \in \Delta(v)$
        if($f$ == STUCK-AT-1)
            count = positive_count(gate driving fault site $f$);
        else
            count = negative_count(gate driving fault site $f$);
        if(count != 0) Detect($f$) += count;
        else NoDetect($f$) += 1;
    last_fail_vec = $v$;

/* Last set of Passing Vectors */
if(All Failures Supplied)
    For passing vectors $p$ = last_fail_vec+1, ..., last_vec
        For faults $f \in \Delta(p)$
            NoDetect($f$) += 1;

/* Calculate Merit values and suspect fault list */
For all faults $f \in F$
    Merit(f) = $C_1 \star$ (NFO - Detect($f$)) + $C_2 \star$ NoDetect($f$);
susp_flt_list = $\{f \in F \mid$ Merit($f$) $<= threshold\}$
```

Figure 3: Fault Localization Algorithm

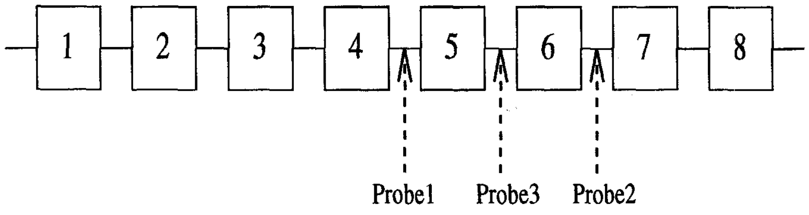

Figure 4: Example of probing scan chain to detect fault in scan chain

chain is roughly 1800. The design has roughly 70,000 equivalent fault classes and 1528 scan test vectors were generated. The fault coverage was 89%. The most of the untestable faults were due to memory modules.

We injected faults on four working parts of this design. The results on those parts as described below.

1. **Part 1** - *A signal line in combinational logic is shorted to VDD.* The tool found the fault site (the net) and the type of the fault (stuck-at-1) in the first run itself. The suspect fault list with single fault assumption (i.e. threshold = 0) contained *only* that fault. Hence, the resolution was also perfect.

2. **Part 2** - *A signal line was shorted to ground.* The net was connected to the output pin of a flip-flop. Test failed in the scan check pattern itself, and it was correctly diagnosed as a fault in scan chain.

3. **Part 3** - *A signal line was cut.* The tool found the fault immediately. The defect behaved like a stuck-at-1 fault at that net. The suspect fault list with single fault assumption (i.e. threshold = 0) contained only 3 faults, including the actual fault site. We also generated diagnostic test patterns for the fault site and it was verified by using electron beam probing that the net indeed behaved like a stuck-at-1.

4. **Part 4** - *Two signal lines were shorted.* This tool was not supposed to handle bridging fault, but we wanted to test the tool for this kind of defects. This fault does not behave as a single stuck-at fault, and the suspect fault list was empty for single fault assumption. By increasing the *threshold* value with C_1 and C_2 set to 1, we generated a suspect fault list of 5 faults. During generation of suspect fault list, we did not have knowledge of the type of the injected fault. It later turned out that the suspect fault list contained one of the two nets involved in bridging, the other net was not in the list. Later we analyzed that the bridging fault was behaving like a stuck-at fault at one of the net for most of the test vectors. That was the reason we could catch this fault. But we may not be as lucky for other bridging faults.

5.2 Test Case 2

A design for Company B is chosen as the second test case for this tool. This design has roughly 30,000 gates. This design was implemented on 200K gate array technology of LSI Logic Corporation with 2 layers of metal. The length of scan chain is roughly 950. The design has roughly 40,000 equivalent fault classes and 975 scan test vectors were generated. The fault coverage was 75%. The reason for low fault coverage is the presence of many black-boxed elements and memory elements.

For this test case, the tool was tested on *real manufacturing defects.* Three defective parts were chosen and the tool was applied on those three parts. The result on those parts are described below.

1. **Part 1** - On this part, the tool produced suspect fault list containing exactly one equivalent fault class. When the defective part was probed, the defect location was found to be exactly at the same place as it was specified by the tool. Hence, the defect localization as well as the resolution was perfect.

2. **Part 2** - On this part, the tool declared that the defect lies in the scan chain itself. Later, it was found by E-beam probing that the defect location is indeed in the scan chain itself.

3. **Part 3** - On this part, the tool could not find any single fault which matched exactly the behavior of this part. The tool was even unable to find any fault which matches closely with the behavior of the defective part. This part was not probed in details to determine the actual defect location due to lack of resources.

6 Conclusions

We have presented a complete system for failure analysis of full-scan circuits. We used a novel scheme of ranking all the possible stuck-at faults in the design according to their merit values, which measures how close the erroneous behavior of the defective part matches with the expected faulty behavior with one of those faults present. This scheme used a heuristic which can handle *multiple faults* up to a certain extent. The user can interactively change the suspect fault list by changing some parameters. If the suspect fault list is big, a new test pattern generation scheme is proposed to distinguish the faults in the suspect fault list. The user can iterate over a defective part until the suspect fault list is small enough. Each suspect fault site is then probed using an E-beam prober to identify the actual defect site and the tool generates a pair of diagnostic patterns to help probing the fault site. This system is developed as a part of C-MDE^{TM} design environment of LSI Logic Corporation and has been applied on several test cases which produced good results.

This system has some limitations. This system is based on stuck-at fault model. Hence, it will not be able to diagnose any defect which does not behave like a stuck-at fault.

References

[1] H. Y. Chang, E. Manning, and G. Metze, "Fault Diagnosis of Digital Systems," *New York: Wiley Interscience*, 1970.

[2] J. A. Waicukauski and E. Lindbloom, "Failure Analysis of Structured VLSI," *IEEE Design and Test of Computers*, pp. 49–60, November 1989.

[3] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital System Testing and Testable Design.* Computer Science Press, New York, 1990.

[4] P. G. Ryan and W. K. Fuchs, "Addressing the Size Problem in Fault Dictionary," *International Symposium for Testing and Failure Analysis*, pp. 129–133, 1993.

[5] I. Pomeranz and S. M. Reddy, "On the Generation of Small Dictionaries for Fault Location," *Internal Conference on Computer-Aided Design*, pp. 272–279, 1992.

[6] P. Camurati, D. Medina, P. Prinetto, and M. S. Reorda, "A Diagnostic Test Pattern Generation," *International Test Conference*, pp. 52–58, 1990.

[7] H. Cox and J. Rajski, "A Method of Fault Analysis for Test Generation and Fault Diagnosis," *IEEE Transactions on Computer-Aided Design*, pp. 813–833, July 1988.

[8] T. Gruning, U. Mahlstedt, and H. Koopmeiners, "DIATEST: A Fast Diagnostic Test Pattern Generator for Combinational Circuits," *International Conference on Computer-Aided Design*, pp. 194–197, 1991.

[9] S. D. Millman, E. J. McCluskey, and J. M. Acken, "Diagnosing CMOS Bridging Faults with Stuck-at Fault Dictionaries," *International Test Conference*, pp. 860–870, 1990.

[10] P. Ryan, S. Rawat, and W. K. Fuchs, "Two-stage Fault Location," *International Test Conference*, pp. 963–968, 1991.

[11] J. Richman and K. R. Bowden, "The Modern Fault Dictionary," *International Test Conference*, pp. 696–702, 1985.

[12] R. C. Aitken, "Fault Location with Current Monitoring," *International Test Conference*, pp. 623–632, 1991.

[13] R. C. Aitken, "A comparison of Defect Models for Fault Location with Iddq Measurements," *International Test Conference*, pp. 778–787, 1992.

[14] R. P. Kunda, "Fault Location in Full-Scan Design," *International Symposium for Testing and Failure Analysis*, pp. 121–126, 1993.

[15] I. Pomeranz and S. M. Reddy, "On Error Correction in Macro-Based Circuits," *International Conference on Computer-Aided Design*, pp. 568–575, 1994.

[16] G. Lindberg, S. Prasad, K. De, and A. Gunda, "Defect Isolation using Scan-path testing and electron beam probing in multi-level high density ASICs," *Microelectronic Manufacturing '94*, 1994.

[17] G. Lindberg, S. Prasad, K. De, and A. Gunda, "'Failure Analysis of Multi-level High Density ASICs using Scan-path testing and Electron-beam Probing," *International Symposium for Testing and Failure Analysis*, 1994.

[18] T. Tamana and N. Kuji, "Integrating Electron-Beam System into VLSI Fault Diagnosis," *IEEE Design and Test*, pp. 23–29, August 1986.

[19] T. Yano and H. Okamoto, "Fast Fault Diagnostic Method Using Fault Dictionary for Electron Beam Tester," *International Test Conference*, pp. 561–565, 1987.

[20] M. Melgara and et al., "Automatic Location of IC Design Errors Using and E-beam System," *International Test Conference*, pp. 898–907, 1988.

[21] N. Kuji and M. Matsumoto, "Marginal Fault Diagnosis Based on E-beam Static Fault Imaging with CAD Interface," *International Test Conference*, pp. 1049–1054, 1990.