A Fault Diagnosis Methodology for the UltraSPARC^{TM}-I Microprocessor

Sridhar Narayanan, Rajagopalan Srinivasan, Ramachandra P. Kunda,
Marc E. Levitt and Saied Bozorgui-Nesbat
Sun Microsystems Inc.
Mountain View, California, U.S.A.

Abstract

In this paper we study the use of precomputed fault dictionaries to diagnose stuck-at and bridging defects in the UltraSPARC^{TM}-I processor. In constructing the dictionary we analyze the effect of the dictionary format on parameters such as memory size, computational effort, and diagnostic resolution. The dictionary is built based on modeled stuck-at faults. However to effectively diagnose both stuck-at and bridging faults, we employ a novel procedure that combines dictionary information with potential bridge defects extracted from layout. Experiments with failing devices show excellent correlation of predicted errors with actual defects.

1 Introduction

Accurate fault diagnosis coupled with failure analysis is an invaluable aid to process improvement. Both recurring and random defects can be identified to improve yield and decrease DPM (defects per million). Diagnosing defects in current state-of-the-art designs proves to be extremely difficult for a number of reasons. These include reduced observability of the huge number of integrated devices, varying logic and circuit design styles, increased clock frequencies, and sophisticated packaging technologies. Diagnosis is further hindered by imprecise knowledge of the possible defects, as well as ways to accurately model such defects. This becomes all the more relevant in migrating to smaller features sizes.

Currently there exist a number of different ways to analyze failing parts. Most of these rely on some form of physical probing. Using physically invasive techniques such as guided-probe or E-beam testing could inadvertently damage a device. To effectively exploit physical methods in failure analysis, it is imperative to provide apriori knowledge of potential faulty locations on a device. In this regard, an automated diagnosis procedure to provide information on fault locales can prove useful. By analyzing failures observed on a tester and comparing results with simulated fault dictionaries, feedback can be provided to identify root causality of failures. The *focus* of this paper is on developing a diagnosis procedure for a current leading-edge design, the UltraSPARC-I processor, based on static dictionary-based methods.

For a design as complex as a microprocessor, fault simulation time and memory usage are constraints in the use of fault dictionaries. In [1], a dynamic diagnosis approach is used to reduce the computation and storage costs associated with fault dictionaries. A comparison is drawn between the use of pre-computed dictionaries and the use of dynamic diagnosis in [2]. Their results show that dictionaries are invaluable to the diagnosis of multiple copies of a circuit. In [3], different techniques in creating fault dictionaries are compared based on parameters such as dictionary size, fault simulation costs and dictionary resolution. In a similar vein, we compare three dictionary formats for stuck-at faults in the UltraSPARC-I to illustrate tradeoffs between dictionary size and diagnostic resolution.

The goal of any diagnosis methodology is to ferret out "real" defects and failure mechanisms. The issue of modeling faults to reflect realistic defects has been studied extensively [4, 5, 6, 7]. In [5], the use of defect classes instead of fault models is suggested to more effectively target the test strategy to defect properties. In our work such an approach was not pursued due to the computational complexity, and inherent limitations of our simulation tools. Our intent is to identify defect locations by employing known fault models and coupling the diagnosis process with layout information.

In [8], a new technique for using stuck-at fault dictionaries for diagnosing bridging faults is described. No specific layout information however is used. In this paper, we employ a similar technique as in [8] to extend the diagnostic capability of our stuck-at fault dictionary and combine this with extracted layout information. This reduces the briding fault search space and improves diagnostic efficiency.

The rest of this paper is organized as follows. To appreciate certain aspects of the methodology, some key design and test features in the UltraSPARC-I are described in Section 2. Detailed descriptions of the

ED&TC '97 on CD-ROM

Device Count	5.2 M (3.4 M in logic)
Die size	15.5mm x 15.5mm
Operating Frequency	167 & 200 MHz
Supply Voltage	3.3V
Technology	CMOS, 4-layer metal
Gate size (L_{drawn})	0.45um
Package	520 pin plastic BGA

Table 1: Statistics on the UltraSPARC-I processor

fault models, dictionary formats and fault rankings are provided in Section 3. We conclude with a flowchart of the diagnosis, some experimental results, and ongoing enhancements to the methodology.

2 UltraSPARC-I Characteristics

For any complex design, the diagnosis methodology is linked to factors such as the fabrication and packaging processes, modes of testing and test patterns, built-in test and debug hardware features, and underlying logic and circuit design styles. A detailed look at the test and debug features of the UltraSPARC-I processor can be found in [9]. In this section we briefly describe the impact of design styles and testability features of the processor on the diagnosis methodology. Some of these features aid effective diagnosis, yet others place constraints on diagnostic resolution.

2.1 Design & Test Features

The UltraSPARC-I processor is Sun's first microprocessor to implement the 64-bit SPARC V9 architecture. Some relevant statistics of the UltraSPARC-I processor, reproduced from [9], are given in Table 1. The design of a majority of the logic blocks are of datapath and standard-cell styles. For such blocks, both the circuit implementation and physical layout relied extensively on automated design tools. In contrast, embedded memory arrays such as SRAMs, TLBs, and register files as well as a number of dynamic logic megacells were custom designed to optimize on area and performance. The methodology in this paper does not address the diagnosis of defects in embedded memory arrays. The test and diagnosis of arrays is done via a dedicated test mode built into the architecture of the processor [9]. The use of four metal layers makes it difficult to acquire signals at lower metal and poly layers. Special probe pads were built into the design to provide observability of key signals with E-beam probing. The number of such probe pads however was greatly restricted by the dense routing of power, ground and clock signals in the upper metal layers.

The UltraSPARC-I design is a near-to-full scan design with approximately 22,000 state elements working under a single clock domain [9]. The ability to scan nearly all flip-flops in the design proves to be an important asset for diagnosis. In essence this allows diagnosis to be performed on a combinational circuit, and greatly simplifies fault simulation and dictionary generation. Diagnosis of any faults in the scan chain are aided by features such as the use of probe pads at different points along the chain, and the capability to break the single chain into multiple chains. In addition, portions of the scan chain can both be initialized and observed via functional logic and non-scan I/O ports.

A number of work have described the effectiveness of using IDDQ to diagnose errors [10, 11]. Although IDDQ-based testing is part of the test flow [9], there exists certain limitations in its use for diagnosis. The power-ground planes for all logic blocks on the processor including the embedded arrays are not independent. No built-in current sensors were included in any blocks to monitor their respective IDDQ currents [11]. Fault effect propagation to the supply rails in dynamic logic blocks is hindered by gated clocks. The most severe constraint to IDDQ-based diagnosis is the magnitude of background leakage on a per-transistor basis and hence cumulatively over all 5.2 million transistors. This leads to quiescent current greater than tens of microamperes and would require a gross defect to detect a perceptible change.

As a first-order means of identifying defective areas in a device, functional tests can prove useful. Most of them are created with the intent of targetting particular design blocks, and exercising specific functionality in them. However as a means of diagnosing specific defect locations, functional tests were not pursued as an option. Accurately simulating fault models with millions of functional test cycles is expensive both in terms of computation time and memory. Use of functional tests for diagnosis, especially in highly sequential designs, also leads to corruption of internal states. The cumulative effect of this corruption might invalidate useful diagnostic information beyond the first failure seen at the pins [4]. The UltraSPARC-I processor has a built-in debug feature that permits the state of the design to be frozen at any given cycle and dumped out via the scan chain. This provides a way to decouple vectors in a functional test sequence to provide more information during diagnosis.

2.2 Fault Equivalence

The degree of accuracy to which modeled stuck-at faults can be distinguished by any diagnosis methodology is limited by *functional equivalence* [12]. The partition of all possible faults into distinct sets of functionally equivalent classes defines the maximal fault resolution. This is an intrinsic aspect of the design being diagnosed. Analysis of the logic blocks in the UltraSPARC-I processor shows that the size of functionally equivalent classes partially correlates with the underlying circuit design style. In particular, class sizes were found to be greater in complex custom logic blocks. As an example, consider the ATPG circuit model shown in Fig 1 for a portion of a custom logic block in the processor. Stuck-at faults (either 1 or 0)

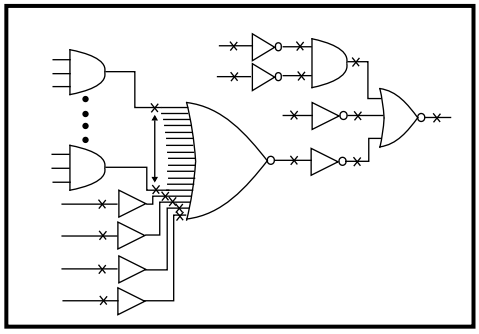

Figure 1: Fault Equivalence Class - Example

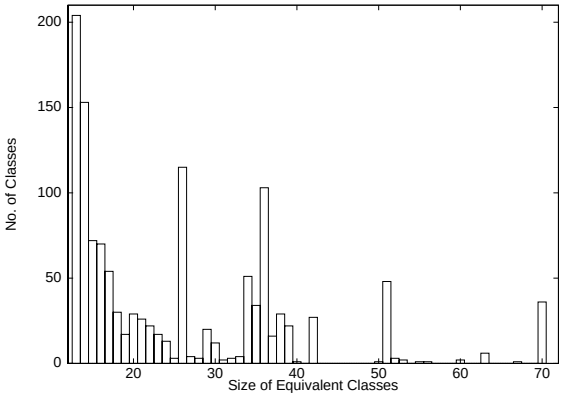

Figure 2: Distribution of Fault Equivalence Classes

at all nets marked "x" are equivalent, leading to a total of 32 equivalent faults. The main contributor to the size of this class is the 18-bit wide NOR gate designed in dynamic logic with a single precharge path and multiple evaluation stacks. The equivalent faults in the portion of logic shown in Fig. 1 are distributed in an area of approximately $125 \times 45 \mu m^2$.

The structurally equivalent classes for all detectable stuck-at faults are derived within the Sunrise ATPG tool [13]. These classes are obtained based on structural topology and gate functionality, and hence only represent lower bounds on the true equivalent class sizes. The distribution of class sizes greater than 10 is shown in Fig 2 for detectable stuck-at faults in the processor. In general the existence of fault class sizes greater than 40 were found to be mainly present in custom-designed blocks, as opposed to blocks where synthesis or structured datapaths were employed. In subsequent discussion, the use of the term fault is synonymous with its fault equivalence class.

3 Diagnosis Methodology

A fault dictionary stores information about errors that a circuit's modeled faults are expected to cause at one or more outputs for each test pattern. It is generated through fault simulation either with or without fault dropping. Defects are located in actual devices by comparing errors observed on a faulty chip to those pre-computed in the dictionary. Based on this comparison, a ranking of the likely modeled faults that could explain the behavior of the faulty part is generated. Further analysis in diagnosing the failure will use the ranking along with physically invasive methods to confirm the existence of the defect.

In using pre-computed fault dictionaries as part of the diagnosis, three main aspects were evaluated and analyzed. These are (a) the interaction between defects and modeled faults used in dictionary generation, (b) the amount of information stored in the dictionary, and (c) the comparison function used in generating the ranking of "most" likely faults. We discuss each of these factors in the following subsections.

3.1 Defects and Modeled Faults

The fault model used to predict defect behavior is an integral aspect of any diagnosis methodology. Fault models provide a way to abstract the behavior of defects, and are essential in providing quantitative measures during test generation, fault simulation and dictionary generation. However fault models are only useful if linked to real physical defects. The most ubiquitous fault model is the stuck-at fault, which is the primary fault model used in this work. Defects that cause stuck-at behavior in CMOS designs include bridging shorts to VDD/VSS, certain forms of gate oxide shorts [10], open defects at transistor gate inputs, and certain open defects that clamp nodes to a supply voltage [5].

The more common defect mode in CMOS designs is the bridging defect caused by unintended shorts between two or more circuit nodes. The bridging defect captures failure mechanisms such as gate oxide shorts, transistor punch-throughs, leaky PN junctions, and physical bridge shorts caused by particles and patterning defects. The most important factor in both the detection and diagnosis of such defects is the *bridge defect resistance*. Experience from previous studies suggests that detection of bridges with voltage tests requires a lower bridge resistance as compared to IDDQ tests [5]. Our diagnosis methodology attempts to use stuck-at fault voltage tests to diagnose potentially low resistance bridge defects.

The option of considering every possible bridging fault in the processor is computationally prohibitive. To prune the size of the fault universe, only bridge defects identified from the layout are considered. Pairs of nodes are individually extracted from all four metal layers of the processor. An extraction flow outputs all signal pairs in the same layer that have edges within $2\mu m$ of each other. These signal pairs are directly mapped to their respective nodes at the switch level of the circuit hierarchy. Note that extracting on a per-layer basis discounts possible inter-layer bridge defects which have a lower probability of occurrence.

Our fault simulator works at the logic gate hierarchy, and was not equipped to simulate bridging faults

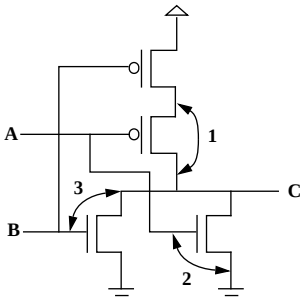

Figure 3: Mapping Layout-Based Bridge Defects

at this level of the design. It was thus essential to devise a technique to (a) map the extracted bridge defects at the switch level to corresponding faults at the gate level, and (b) employ stuck-at fault simulation to generate a dictionary for bridging defects. To address the first of the above, a heuristic procedure was developed to browse the switch-level design hierarchy and map nodes involved in a bridge to corresponding nodes at the gate-level. Nodes that have a one-to-one correspondence with gate inputs/outputs are easily mapped. The difficulty arises when dealing with intra-node bridges within a gate. As an example, consider the 2-input NOR gate shown in Figure 3. Bridge defects 3 and 2 are mapped into a bridge between input B and output C, and a stuck-at-0 at input A, respectively. However, since one of the nodes involved in bridge defect 1 has no mapping to a gate-level node, the bridge defect is neglected. Bridge defect coverage is lost in this fashion for certain complex gates as well as for some switch-level macros such as transmission gates.

To provide an idea of the loss in bridge coverage, the following percentage of bridges were obtained in analyzing metal layer 4. A unique sort of the extracted bridges reduced the number to 50%. The extraction flow outputs multiple instances of a particular bridge if the two nets remain adjacent for lengths greater than $3\mu m$. The large number of such multiple bridge instances reflects the extensive power/ground routing in metal layer 4. Removal of bridges that could not be mapped to gate-level nodes further reduced the bridge defect percentage to 48.9%. Finally bridges to VDD/VSS, that potentially manifest as stuck-at faults, are removed leading to 26.5% of gate-level bridge defects. Similar percentages for metal layer 3 were evaluated to be 20% on the unique sort; 19.63% after mapping to gate-level nodes; and 16.73% after removal of shorts to VDD/VSS.

Having generated a realistic bridging fault universe at the logic gate level, the next step was to provide a way to diagnose these bridging faults using the chosen set of ATPG patterns. One option would have been to fault simulate each bridge to build a corresponding dictionary for bridging faults. Since our fault simulation tools were not equipped to simulate bridging faults, we adopted a technique that uses information from a stuck-at fault dictionary. The information stored in a stuck-at fault dictionary can be used to accurately diagnose bridging defects. As outlined in [8], the basic principle used is that any pattern detecting a bridging fault must also detect a stuck-at fault on one of the shorted nodes.

3.2 Dictionary Format

The degree of information stored in a dictionary is a tradeoff between computation time, storage and diagnostic resolution. To study these tradeoffs for detectable stuck-at faults in the UltraSPARC-I processor, three different dictionary formats are analyzed. Any such comparison is influenced by the set of test patterns used in creating the dictionaries. Different sets of test patterns will yield varying results. We use a single test set to highlight relative measures of the tradeoffs for three dictionary formats. The chosen set of ATPG patterns were the ones used as part of the production test flow for the processor.

The three dictionary organizations are the *complete*, *vector-based* and *frequency-based* formats. These three formats are illustrated in Fig. 4 for a hypothetical circuit of three outputs and four faults (equivalence classes) being tested with two patterns. The complete dictionary can be viewed as a matrix of $\mathcal{F}$ rows and $\mathcal{V} \times \mathcal{O}$ columns, where $\mathcal{F}$ denotes the number of faults, $\mathcal{V}$ denotes the number of test patterns, and $\mathcal{O}$ denotes the number of primary/scan outputs. As shown in Fig. 4(a), a bit in the matrix identifies whether the fault is detected at a particular output for the given pattern. Note that generation of a complete dictionary requires fault simulation with no fault dropping.

The vector-based dictionary adopts a more gross resolution in only specifying whether a given pattern detects a fault at one or more outputs. It can be viewed as a matrix of $\mathcal{F}$ rows and $\mathcal{V}$ columns. Finally the frequency-based dictionary is a one-dimensional matrix in which each fault is associated with a frequency count. This count denotes the number of times the fault is detected at one or more outputs for all patterns. From Fig. 4, one can compare the level of information stored under each format. To provide a quantifiable measure of diagnostic resolution for the three formats, we make use of the following definitions.

A *diagnostic class* represents a set of faults that a particular dictionary cannot distinguish. The number and size of diagnostic classes for a particular dictionary depends on the test set used in generating the dictionary. *Diagnostic power* for a limit $\mathcal{K}$ is defined as the fraction of all faults that lie in diagnostic classes of size $\mathcal{K}$ or less. The *diagnostic expectation* is defined as the simple average of diagnostic class sizes over all faults; this correlates to the average size of a fault list produced by a diagnosis when all faults are equally

	V_1			V_2		
	O_1	O_2	O_3	O_1	O_2	O_3
F_1	1	0	0	0	1	1
F_2	0	1	1	1	0	1
F_3	0	1	1	0	0	0
F_4	0	0	0	1	0	1

(a)

	V_1	V_2
F_1	1	1
F_2	1	1
F_3	1	0
F_4	0	1

(b)

	Freq
F_1	3
F_2	4
F_3	2
F_4	2

(c)

Figure 4: Dictionary Formats: (a) complete (b) vector-based (c) frequency-based

Dictionary Format	Dictionary Size (MB)	Diagnostic Expectation	Maximal Class Size
Complete	638.8	1.95	233
Vector	96.7	21.11	1444
Frequency	7.9	4194.63	16817

Table 2: Comparing diferent Dictionary Formats

likely [3].

In Table 2, we compare the three dictionary formats based on dictionary size, diagnostic expectation, and maximal size of a diagnostic class in each format. A more effective comparison can be drawn from Fig. 5 in which the diagnostic power is plotted against the size of diagnostic classes. Both the complete and vector-based dictionaries achieve 80% diagnostic power for class sizes less than 5, with the former approaching 100% within a size of 20. The frequency-based dictionary clearly provides the least diagnostic resolution. Note the six-fold increase in the size of the complete dictionary as compared to the vector-based one.

In spite of its larger memory requirements, the complete dictionary was used in our methodology as opposed to the vector-based one. The advantages in doing so are two-fold; in addition to providing higher stuck-at diagnostic resolution, retaining as much information in the dictionary is important to its dual use in diagnosing bridging faults. Comparisons with a dictionary built on *drop-on-k* fault simulation strategies are currently being investigated.

With the complete dictionary for stuck-at faults, a secondary dictionary based on the bridging fault universe is generated. In this dictionary a *composite signature* is associated with each bridging fault. The composite signature is the union of the signatures of the four stuck-at faults associated with the bridge. In the example of Fig. 4, if we assume that the four faults are the respective stuck-at-1 and 0 of the two nodes in a bridge, the composite signature is given by a row of all 1's in the bridge fault dictionary. In the absence of oscillations, the use of composite signatures does guarantee that if a bridging defect is the failure mode, at least one of the nodes involved in the bridge will be identified in the diagnosis [8].

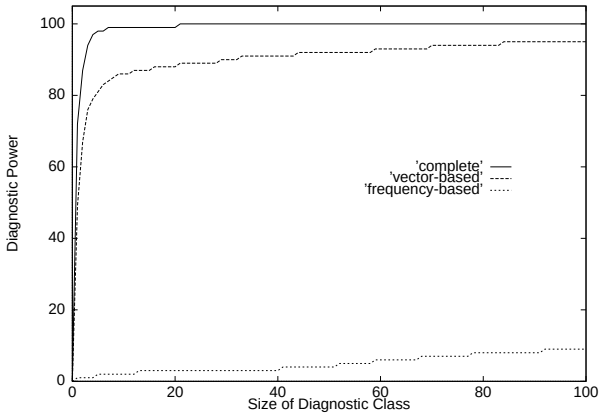

Figure 5: Comparing Dictionary Formats

3.3 Ranking the Faults

In the case where the response of a faulty circuit cannot be solely explained by a single modeled stuck-at or bridging fault, a technique is required to rank the possible faults in order of decreasing likelihood. An additional requirement in our methodology is to synergize diagnostic information from both the stuck-at and bridging fault dictionaries. All stuck-at and bridging faults in their respective dictionaries are assumed to be of equal likelihood.

The input to the diagnosis procedure consists of the failing patterns with the respective scan/primary output bits at which mismatches are detected. Let α denote the total number of failing output bits over all patterns. Let us first consider the diagnosis procedure with the stuck-at fault dictionary. The stuck-at fault dictionary is stored as a set of files, where each file corresponds to an ATPG test pattern. Within a file, each output bit is associated with a set of stuck-at faults detected at that output. Initially all faults are assigned an *intersection count* of zero. For a given input to the procedure, only those files corresponding to the failing patterns are accessed. The intersection count of a fault is incremented each time it is associated with a mismatching output bit in any of these accessed files. The output of the procedure is a ranking of faults in decreasing intersection counts. This ranking scheme is based on the partial intersection technique presented in [14]. Note that this scheme only uses failing output bits and does not utilize any information on the passing patterns. The intersection count of the fault ranked highest is thus upper bounded by α.

The bridging fault dictionary is stored as a single file of "bridging" faults where each fault is associated with a set of *(pattern, bit)* pairs. The set of *(pattern, bit)* pairs represent the composite signature derived from the four stuck-at faults associated with the bridge. The intersection count of a bridging fault corresponds to the number of *(pattern, bit)* pairs in the intersection of its set with the input failing patterns/bits. The intersection count of any bridg-

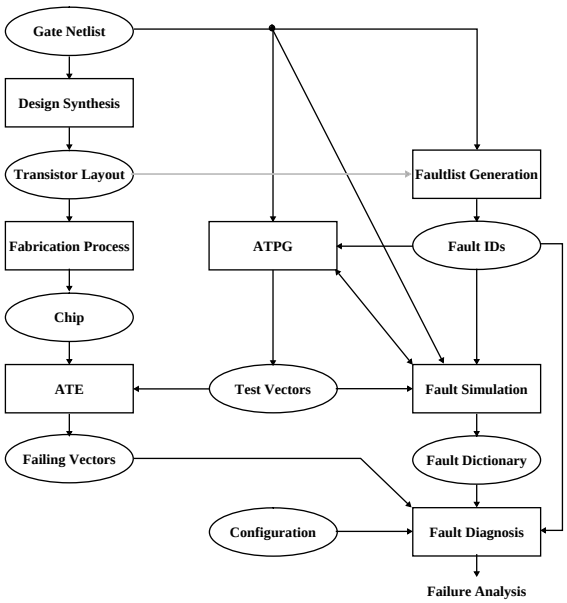

Figure 6: Diagnosis Methodology Flowchart

ing fault is also upper bounded by α. Similar to the stuck-at fault case, the bridging faults are ranked in decreasing intersection counts.

The highest ranked faults in both the stuck-at and bridging lists are ideal candidates for failure analysis. Manual inspection of both lists however is required to decide on which locations to inspect. Let β and γ represent the intersection counts of the highest ranked stuck-at and bridging fault(s), respectively. If either β or γ is equal to α, the corresponding fault(s) is most likely the potential defect. In this case, the size of the list for failure analysis is equal to the number of faults with count α. For cases when both β and γ are less than α, the fault rankings need to be analyzed to derive an appropriate list for failure analysis. This is empirically done based on the intersection counts of the top ranking faults, as well as the composition of the stuck-at faults in the top ranking bridging faults. For example, if the intersection count of the top ranked bridging fault is solely due to one of its four stuck-at faults, β will be greater than or equal to γ. In such cases the top ranking stuck-at faults will mainly be used in failure analysis.

4 Experimental Results

The flowchart for the complete fault diagnosis methodology is shown in Figure 6. All single stuck-at faults, collapsed at the gate netlist level, are enumerated by the faultlist generator. Each single stuck-at fault is represented by a unique fault ID number in the database. The bridging fault universe consists of all pairs of adjacent layout nets as outlined in Section 3.1. Each bridging fault is represented by a set of four fault IDs corresponding to the four stuck-at faults on the bridged nets.

To create both dictionaries, test patterns are deterministically generated for the stuck-at fault model using the Sunrise ATPG [13]. Failing patterns from devices tested either with automatic test equipment (ATE), or at JTAG-based workstation environments are used in the diagnosis. The diagnosis program is directed by means of a configuration file which contains information about the design netlist, fault dictionary, test patterns and a list of options invoked during the analysis.

4.1 Validation and Use on Failing Parts

To validate the methodology, defects were induced in a number of functional UltraSPARC-I parts. These defects were introduced via focussed ion beam milling (FIBs) and consisted of shorts to VDD/VSS, interconnect bridges as well as "blind" defects. For the blind defects, the diagnosis procedure had no apriori knowledge of the location and nature of the induced defect. Both feedback as well as non-feedback bridges were introduced. Table 3 shows results for seven parts.

In the table the first two columns show the induced fault and the metal layer on which it was introduced. For the case of the two blind defects, the metal layers were identified as a result of the diagnosis. The CPU times and memory usage for both the stuck-at and bridging fault diagnosis are shown. Note that the memory utilization for the bridging diagnosis is independent of the failure mode of the device. The composite signatures of all bridging faults are compared with the input failing patterns/bits, unlike the stuck-at fault dictionary where only the required files are accessed. The α, β and γ columns are based on the terminology in Section 3.3. The size of list shows the number of faults that would be sent to failure analysis based on evaluating the rankings.

The size of the list only refers to fault equivalence classes; the faults within each equivalence class are stored in our database and provided to failure analysis. For the case of part 2, note the large size of the list. The diagnosis identified both a stuck-at-0 and a stuck-at-1 on a particular net, and the bridging fault universe had seven other nets that were adjacent to this net. In every other case, the size of the list produced was less than 4. Both the blind defects were bridging faults since γ was greater than β. In all cases the induced defect was among the faults provided by the diagnosis.

In Table 4 we show results on four failing parts obtained from our vendor. The format is similar to that in Table 3. The observed defects in all four parts were due to particles in different metal layers. All of them were uncovered by exclusively looking at the locations provided by the diagnosis procedure. Current work is ongoing to evaluate the diagnosis procedure for other failure modes.

No	Induced Fault	Metal Layer	Count			Time (sec)		Memory (KB)		Size of List
			α	β	γ	SF	BF	SF	BF	
1	Stuck-at-1	M4	204	204	204	299	1354	9	43000	2
2	Bridging	M4	354	305	354	533	1905	9	43000	7
3	Stuck-at-0	M3	123	122	123	332	1115	12	43000	3
4	Bridging	M3	116	116	116	39	1052	4	43000	3
5	Bridging	M2	45	45	45	77	780	5	43000	2
6	Blind	M3	77	30	77	103	912	5	43000	1
7	Blind	M3	18	10	18	26	698	3	43000	2

Table 3: Validation Results on FIBed UltraSPARC-I Devices

No	Count			Time (sec)		Memory (KB)		Observed Defect	Size of List
	α	β	γ	SF	BF	SF	BF		
1	37	37	0	104	767	12	43000	Small metal particle	1
2	6	5	3	48	645	4	43000	Small loop particle	3
3	469	291	17	492	2338	13	43000	Small metal particle	3
4	425	100	122	715	2213	20	43000	Small metal particle	5

Table 4: Diagnosis Results on Failing UltraSPARC-I Devices

5 Conclusion

The diagnosis methodology has been effective as an aid to failure analysis, and is currently being deployed on other processors. Using both the stuck-at and bridging dictionaries enhances diagnosis when targetting realistic defects on devices. The use of full scan was an invaluable asset in the efficient generation and use of a complete dictionary with high diagnostic resolution.

A number of features are currently being evaluated to enhance the methodology. One aspect alluded to in Section 3 is the effect of different test sets. The test set used in our work was an optimized set of patterns targetted to detecting stuck-at faults. The use of random test sets, uncompacted stuck-at tests, and bridging fault test patterns needs to be investigated. In addition ways to employ functional test sequences in diagnosis are being evaluated. The methodology outlined in this paper follows a *cause-effect* based analysis [12]. Some effect-cause analysis to prune the list of potential faults will further increase diagnostic efficiency. This could be in the form of circuit cone analysis and/or generation of diagnostic patterns targetting specific defects.

6 Acknowledgements

The authors would like to thank the following individuals: Kenneth Ho, Dave Maxwell, Ganesan Vidyasagar and Ron Melanson from SUN Microsystems, and Karl Johnson and Mark Nolen from Texas Instruments.

References

[1] P. G. Ryan, S. Rawat, and W. K. Fuchs. Two-Stage Fault Location. In *Proc. IEEE Int'l Test Conf.*, pages 963–968, October 1991.

[2] I. Pomeranz and S. M. Reddy. On the Generation of Small Dictionaries for Fault Location. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 272–279, November 1992.

[3] P. G. Ryan, W. K. Fuchs, and I. Pomeranz. Fault Dictionary Compression and Equivalence Class Computation for Sequential Circuits. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 508–511, November 1993.

[4] R. C. Aitken. Finding Defects with Fault Models. In *Proc. IEEE Int'l Test Conf.*, pages 498–505, October 1995.

[5] C. F. Hawkins et al. Defect Classes - An Overdue Paradigm for CMOS IC Testing. In *Proc., IEEE Int'l Test Conf.*, pages 413–425, October 1994.

[6] A. Jee and E. J. Ferguson. CARAFE: An Inductive Fault Analysis Tool for CMOS VLSI Circuits. In *Proc. IEEE VLSI Test Symp.*, pages 92–98, April 1993.

[7] W. Maly. Realistic Fault Modeling for VLSI Testing. In *Proc. Design Automation Conf.*, pages 173–180, June 1987.

[8] S. D. Millman, E. J. McCluskey, and J. M. Acken. Diagnosing CMOS Bridging Faults with Stuck-at Fault Dictionaries. In *Proc. IEEE Int'l Test Conf.*, pages 860–870, October 1990.

[9] M. E. Levitt et al. Testability, Debuggability and Manufacturability Features of the UltraSPARC-I Microprocessor. In *Proc. IEEE Int'l Test Conf.*, pages 157–166, October 1995.

[10] J.M. Soden et al. IDDQ Testing: A Review. *Journal of Electronic Testing: Theory and Applications*, 3(4):291–304, Nov 1992.

[11] W. Maly and M. Patyra. Design of ICs Applying Built-in Current Testing. *Journal of Electronic Testing: Theory and Applications*, 3(4):397–406, Nov 1992.

[12] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design.* IEEE Press, 1990.

[13] Sunrise Reference Manual. Technical report, Sunrise Test Systems, 1996.

[14] R. P. Kunda. Fault Location in Full-Scan Design. In *Proc. Int'l Symp. for Testing and Failure Analysis*, pages 121–126, October 1993.