A Dynamic Diagnostic Test Generation System for I_{DDQ} Measurement Based Diagnosis of Bridging Faults

Yiming Gong and Sreejit Chakravarty
Department of Computer Science
State University of New York
Buffalo, NY 14260

Abstract

A paradigm for diagnosis, known as dynamic diagnosis, is defined. A system based on this paradigm, for I_{DDQ} measurement based diagnosis of bridging faults, is reported. Experimental evaluation of the performance of the system shows it to be substantially superior to existing systems, especially for larger circuits.

1 Introduction

Given a circuit and the observed faulty response of the circuit, diagnosis (or fault-location) is the process of locating physical faults which result in the faulty response. Such analysis is used to gather information to improve the fabrication process.

There are different approaches for fault diagnosis which includes **cause-effect** analysis and **effect-cause** analysis [3]. In cause-effect analysis, before diagnosis, simulation is used to determine the possible responses of the circuit, to a given test sequence, in the presence of the targeted faults. These responses, along with the faults which cause them, are stored in a **fault dictionary**. To locate faults, the same test sequence is applied to the faulty circuit. If the observed faulty response matches one of the responses in the fault dictionary, the corresponding faults in the fault dictionary constitute the possible set of faults in the faulty circuit. This is the prevalent approach being used today.

The tasks that needs to be performed in cause-effect analysis are as follow. A diagnostic test sequence (DTS), targeting a given set of faults F, is precomputed. This DTS is evaluated as to its effectiveness in distinguishing between faults in the set F (**diagnostic simulation** [10, 11, 19, 26, 27]). A fault dictionary is computed either independently or as a by product of diagnostic simulation. This apriori computation of the DTS and fault dictionary is referred to as **static computation**. The statically generated fault dictionary, known as **static fault dictionary**, is used during diagnosis. The statically generated diagnostic test sequence is known as **static diagnostic test sequence**. Instead of using the term *cause-effect* we refer to this process as **static diagnosis**.

Static diagnosis has several disadvantages.

- In spite of considerable progress [10, 21, 26, 31], unlike fault simulation, diagnostic simulation is an expensive process. It requires considerable CPU time and storage space. This is especially true if we move away from the classical stuck-at fault models and consider bridging faults [8].
- Static fault dictionary often requires considerable space thereby leading to maintenance problem. In addition, the time required to compute them is also very large. Much effort has been directed at reducing the required space [5, 23, 27, 28, 29, 30]. These techniques, although very useful for medium sized circuits and stuck-at faults, starts falling apart when larger circuits and fault models with larger number of faults are considered.
- The DTS, unlike tests needed for fault detection, can be pretty large. Although this is not the case for stuck-at faults in small circuits [7, 17, 22] it is a problem for large circuits and fault models with larger number of faults in them [8]. Detection oriented stuck-at test sequences can be used for diagnosis purpose, but they are not good DTSs in the sense that they can not partition faults into small equivalence classes. That is they do not have good diagnostic resolution. To achieve good diagnostic resolution these test sequences need to be augmented. This results in large test sequences. Storing and maintaining these large test sequences poses a problem.

Having listed the drawbacks of static diagnosis we have to put it in proper perspective. Static diagnosis is useful when a large number of faulty chips having different faulty responses need to be diagnosed. The high cost associated with static diagnosis is then amortized over the large number of faulty responses that needs to be analyzed. On the other hand, it is often the case that many of the faulty chips have the same faulty response. Therefore, the number of distinct responses to be analyzed is small. In this case the above disadvantages of static diagnosis become important.

In addition, the above disadvantage are not significant if the fault model does not contain too many faults. However, various studies have high lighted the importance of targeting non-stuck-at faults like leakage, bridging faults, etc.[14, 15]. Bridging fault(BF) occurs when two or more distinct nodes of the circuit are unintentionally connected. Bridging faults model 40% to 50% of physical faults in contemporary MOS technologies [14]. To improve the quality of VLSI chip, analysis of such fault must be addressed. Once we target such fault models the disadvantages of static diagnosis become more significant.

Motivated by this, alternatives to static diagnosis have been studied. One approach uses effect-cause analysis[1, 2, 9, 12, 28]. In this approach no static fault dictionary is computed thereby eliminating the corresponding storage and maintenance problems. In [1, 2, 12] no fault dictionary was computed. In [9] portion of *stuck-at fault* dictionary was computed, "on the fly", for diagnosis of *bridging faults*. A DTS still needs to be computed apriori. In addition, the expensive step of diagnostic fault simulation is still required. Storing and maintaining large test sequences is still a problem. This approach to diagnosis is known as "dynamic diagnosis" in the literature [23]. Since this approach assumes that the DTS is available, a precomputed DTS is still needed, we prefer to use the term **quasi-static diagnosis**.

In this paper we first define an approach to diagnosis, discussed in section 2, which we refer to as **dynamic diagnosis**. Note that the usage of this term is different from its usage in [23]. To our knowledge, such an approach has not been discussed in the literature.

In dynamic diagnosis we *do away with the costly steps of computing static fault dictionaries, diagnostic simulation as well as static diagnostic test generation*. The need to store the *fault dictionary* or *the diagnostic test sequence is eliminated* thereby avoiding the problems stated above.

In dynamic diagnosis the diagnostic test sequence is computed dynamically, with the help of the faulty response, in several phases. In each phase a small test sequence is generated and faults dropped from the fault list. Based on the reduced fault list, in the next phase, a new sequence of tests is generated and applied to the faulty circuit. This procedure continues until we reach the point where all faults in fault list are functionally equivalent making this a "complete" diagnostic test generation scheme. This is known as **dynamic diagnostic test generation(DDTG)**.

We have developed a diagnosis system based on this paradigm. Our system targets BFs between two nodes. It assumes that I_{DDQ} measurement is the testing technique being used.

The system is a blend of several ideas which includes ideas for representing and dropping BFs[12], fault collapsing for BFs, and techniques for exploiting stuck-at test generators for computing diagnostic test sets.

The performance of the system has been evaluated through experimentation using ISCAS85 and full scan versions of ISCAS89 circuits. We have compared the performance of the proposed system with existing systems. The performance of the system was found to be substantially better, especially for the larger circuits.

2 Dynamic Diagnosis

Dynamic diagnosis is defined. It is a dynamic process in that the diagnostic test sequence is generated during diagnosis. It is also a multi-phase, iterative process. The response of the chip under diagnosis(CUD) is fed back to the diagnostic test generator. The diagnostic test generator drops faults and generates a new set of test vectors. The process continues till no fault dropping is possible, i.e., all faults in fault list are equivalent.

Each phase consists of two parts, diagnostic test generation and fault dropping. The two parts interact with each other. In the first part, a small test sequence targeting the remaining faults in the fault list is generated. It is then applied to the CUD. The responses, which could either be the logic levels at POs (logic measurement), or the quiescent power supply current (I_{DDQ} measurement), or a combination of the two, of the chip to the test sequence are fed back to the diagnostic test generator. This information is then used to reduce the fault list (fault dropping). After this the next phase starts with the reduced fault list. This process continues until we reduce the fault list to a set of functionally equivalent faults. In that sense dynamic diagnosis is always "complete" with respect to the targeted faults and the measurements it used.

Figure 1 shows the structure of the diagnosis system required. This system will be used as follows. For each phase, the diagnostic test generator will generate a test set. This test set will be down loaded from the workstation to the tester. The test set will be applied to the CUD. The response recorded and fed back to the diagnostic test generator in the workstation. This system requirement is no different from what is required by the static diagnosis paradigm.

Ideally, in each phase only one test is generated. This makes the diagnostic test generator "well informed" about the status of the diagnosis process. But this implies a huge communication overhead. We therefore generate tests in groups, and each group is processed in one phase. The number of phases and the number of tests in each group depends on fault models and the approaches for fault dropping and test generation. We will discuss the details for one class of faults and testing strategy.

There is a potential problem with this approach to diagnosis. The amount of time the CUD spends in the tester (tester time) may be too large due to the following reasons: the diagnostic test generator spends too much time during one or more of the phases; or the number of phases is too large. Care must be taken to avoid these potential problems. For the case that we study we will show how this can be done very effectively.

3 A Dynamic Diagnosis System For Bridging Faults

A three-phase dynamic diagnosis algorithm for bridging faults is presented. It assumes that I_{DDQ} measurement is being used. Here, as we mentioned in section 2, each *phase* means one iteration of test generation and fault dropping. We first present some background about BFs and I_{DDQ} measurement. Then we present the three-phase dynamic diagnosis system.

3.1 I_{DDQ} Test and Fault Dropping

A BF between two gate outputs can be detected by I_{DDQ} measurement *iff the two shorted lines have different values*[4, 6, 12]. For example, on application of $t_1 = \langle x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1\rangle$ to the circuit of Figure 2, a path from $G5$ to V_{SS} and a path from $V_{DD}$to $G6$ is activated. Gate output $G5$ has logic value 0 and gate output $G6$ has logic value 1. If the BF $\langle G5, G6\rangle$ exists there will be a conducting path from V_{dd} to V_{SS} through the activated paths. The resulting non-zero I_{DDQ}, detected either by an internal or an external device, is used to signal the presence of a BF.

In our diagnostic test generation algorithm we use a compact representation for BFs[12]. BF between lines x, y are represented by $\langle x, y\rangle$. $\{\langle A, B\rangle\}$ represents a set of faults $\{\langle u, v\rangle | u \in A \; and \; v \in B\}$. Representation $\{\langle A, B\rangle, \langle C, D\rangle, \ldots\}$ means $\{\langle A, B\rangle\} \bigcup \{\langle C, D\rangle\} \bigcup \ldots$. For example, $\{\langle\{1, 2, 3\}, \{4, 5\}\rangle\}$ represents the set of faults $\{\langle 1, 4\rangle, \langle 1, 5\rangle, \langle 2, 4\rangle, \langle 2, 5\rangle, \langle 3, 4\rangle, \langle 3, 5\rangle\}$.

Fault dropping for BFs using I_{DDQ} is accomplished as follow [13]. On application of an input vector, the lines (gate outputs) are set to either logic 0 or logic 1. This partitions the lines in the circuit into two set, lines with logic value 0 and lines with logic value 1. If the input vector detects the existing fault, the existing fault must be between lines with different logic values. Faults between lines of same logic value can be dropped from the fault list being processed. We retain the complement of this set if the input vector does not detect the fault. Efficient methods for accomplishing this is discussed in [12, 13]. This idea can be extended to include internal nodes which can take on values 0, 1 or f [13]. Here f denotes a floating value.

3.2 The Dynamic Diagnostic Test Generation System

Let $V(x)$ denote the logic value of line x. Two BFs, $\langle x, y\rangle$ and $\langle u, v\rangle$, are **I_{DDQ} Equivalent** iff $V(x) \neq V(y) \iff V(u) \neq V(v)$. Such faults can not be distinguished using I_{DDQ} measurement alone. The objective of the diagnosis system is to reduce the set of faults to one I_{DDQ} equivalent class.

An overview of our dynamic diagnostic test generation system is given in Figure 3. The three-phase algorithm consists of a phase of **random test generation**, a phase of generating tests **targeting low**

controllability lines, and a phase of **deterministic test generation**. In between, after the phase of targeting low controllability lines, a procedure is used to **identify equivalent faults**. This procedure does not generate any test vectors and hence does not form a "phase". This four-step, three-phase algorithm constitutes a complete diagnosis algorithm. We next describe each step.

3.2.1 Random Test Generation Phase

As mentioned earlier, two faults $\langle x, y\rangle$, $\langle u, v\rangle$ are I_{DDQ} equivalent iff $V(x) \neq V(y) \iff V(u) \neq V(v)$. Assume that lines in a circuit are not correlated with each other and each line can be set to 1 and 0 with equal probability. Then a randomly generated input vector partitions all the lines into two equal parts, half of the lines in the circuit would be set to 0 and half would be set to 1. In other words, half of the BFs would be between lines of different logic values and half of the BFs would be between lines of same logic values. The former gives a large current. The latter does not. By monitoring the current, we can tell these two sets of faults apart. We call this partition **0-1 partition**.

If we further assume that randomly generated input vectors are not correlated, then the 0-1 partitions given by each input vector will not be the same. In the best possible scenario, different input vectors partition the lines in orthogonal directions. Let the total number of lines in a circuit be n. Since each input partitions the lines into two equal parts, each input vector reduces BFs in half. Therefore, in the best possible case, $\log(n)$ input vectors will reduce the number of faults to 1. Therefore, $\log(n)$ is a lower bound on the number of vectors that will reduce the set of faults to one I_{DDQ} equivalent class. Therefore, **in the first phase, an order of $\log(n)$ number of random input vectors are generated**, using equiprobable distribution. Using this test set faults were dropped.

3.2.2 Lines with Low 0(1)-Controllability

The above step will not generate all distinguishing tests for two reasons. Firstly the above assumptions that lines are not correlated is not always true. We will deal with this problem later. Secondly, some lines cannot be set to 1 or 0 with equal probability. If line x is hard to set to 0(1) then we say that x has low 0(1)-controllability[18, 24]. Consider faults $\langle x, y\rangle$, $\langle a, b\rangle$ and assume that each of x, y, a, b have low 0-controllability. Then the probability that a randomly selected test vector will set one of them to 0 and rest of them to 1 is low. The probability that it will set three of them to 0 and only one of them to 1 is even lower. It is highly likely that such faults will not be detected during the random phase. A similar conclusion can be drawn if x, y, a, b have low 1-controllability. Therefore, **in the second phase test vectors which distinguish faults between lines of low 0(1)-controllabilities are generated and applied to reduce the number of faults in fault list**. The controllability of the lines, based on the randomly generated test set, is determined in the first phase in a manner similar to STAFAN[18].

There are a few ways to solve the low controllability problem. Let us still assume that these lines with low controllabilities are not correlated. Then increasing the number of random test vectors will help us distinguish these faults. Alternatively, as is done for detection test generation, we can raise the controllabilities of some of these lines by selectively assigning logic values to some of the PIs and randomly assigning values to the rest. Another way to do it is to bias the random test generator to assign some PIs certain logic values more often than the others.

In our algorithm a different approach is used. For each fault in fault list the 0(1)-controllabilities of the two shorted lines are examined. A set of lines with low 0(1)-controllabilities is identified. A single stuck-at fault (SSF) test generator is used to generate tests to detect those lines stuck at values with low controllability. That is, if line x has low 0-controllability, we will generate test to detect line x stuck at 1. These tests are used as diagnostic tests in phase 2 of our algorithm.

3.2.3 Fault Collapsing

After applying random and possibly biased random test set or the test set we computed and dropping faults from fault list, now we have reason to believe that faults remaining in fault list are highly correlated. Although not all of them are I_{DDQ} equivalent faults, they most likely are functions of each other or functions of some common lines. In next step we try to recognize as many I_{DDQ} equivalent faults from this set of faults. This step does not generate any test vectors. It only reduces the number of distinct faults in fault list.

Figure 4 shows two kinds of I_{DDQ} equivalent faults. In (a) faults $\langle a, d\rangle$ and $\langle c, d\rangle$ are equivalent. We call this type of equivalent faults first type of equivalent faults.

In Figure 4(b) faults $\langle x_i, y\rangle$ and $\langle x_j, y\rangle$, where $x_i, x_j \in \{x_0, x_1, ..., x_{n-1}\}$, y is an arbitrary line in circuit, are I_{DDQ} equivalent if logic functions $F_1 = F_2 = \cdots = F_{n-1}$. There is an analogous generalization for Figure 4(a). In general, finding whether two lines in a given circuit are equivalent or not is NP-Complete[16]. Therefore, heuristics are used to determine such sets[25]. Our heuristic is a generalization of the one used in [25]. Note that the fault equivalence given by Figure 4(a), involving "cross coupled NOT" gates is unique to diagnosis and has not been discussed before.

The heuristic we used is as follow. Assume $\{\langle A, \{v\}\rangle\}$ is a set of faults $\{\langle x, v\rangle | x \in A\}$. If, for two lines x, $y \in A$, and $V(x) = V(y) = f(z)$, where z is a line in the circuit and $f() = \prod_1^k FUN_i() = FUN_1(FUN_2 \cdots FUN_k() \cdots)$, FUN_i is either logic NOT or a buffer, then faults $\langle x, v\rangle$ and $\langle y, v\rangle$ are I_{DDQ} equivalent. And x and y are called **I_{DDQ} equivalent lines**.

To identify second type of equivalent faults, we use breadth first search, with the help of a parity bit, to identify the I_{DDQ} equivalent *lines*. The parity is initially set to *even*. Whenever a NOT gate is encountered, the parity is flipped. Note that if two lines are equivalent they must have same parity. We next give the algorithm for identifying this type of equivalent faults by identifying functionally equivalent lines. It assumes the above fault list format $\langle A, \{v\}\rangle$; a line here is the fanout of a gate; $cgate$ denotes current gate; p denotes current parity; and E is the set of I_{DDQ} equivalent lines identified so far. The procedure ends when either a non-unary function is encountered or the search area is out of set A.

Procedure(Identify_Trivially_Eq_1(cgate))
    Starting with a line in A (cgate) which is closest to POs;
    p = even;
    E = cgate;
    Loop for ever
        IF (cgate != unary function) RETURN(E);
        IF (cgate == NOT) p = $\tilde{p}$*;*
        cgate = fanin gate of cgate;
        IF (p == even && cgate $\in A$*) Append(cgate, E);*
        IF (cgate == fanout point)
            FOR each fanout branch other than where we came
            E1 = Forwardsearch(branch_line, p); // It returns a set of equivalent lines.
        Append(E1, E);
End(Procedure)

Procedure(Forwardsearch(cgate, p))
    IF (cgate != unary function) RETURN(NULL);
    IF (cgate is closer to POs than where we started) RETURN(NULL); // cgate $\notin A$
    IF (cgate == NOT) p = $\tilde{p}$*;*
    IF (p == even && cgate $\in A$*) Append(cgate, E);*
    FOR each fanout branch
        E1 = Forward(branch_line, p);
        Append(E1, E);
    RETURN(E);
End(Procedure)

Example: Given circuit in Figure 5. Assume we have a set of faults $\{\langle\{a, b, c, d, e, f, g, h, i, j, k\}, \{v\}\rangle\}$, we want to identify second type of equivalent faults. Starting from a, we first backtrace to j. Since j is a fanout point, each branch other than a is searched. Lines b, c, d and k are identified as equivalent lines. Then we back trace to f through e. Since f has the same parity as line a has, it is identified as equivalent line, too. f is a fanout point, fanout branch g is searched and eventually we will identify line h and i as equivalent lines. Now the control comes back to line f. Since gate f is not a unary function (NOR gate), we end our search. Lines in $\{a, b, c, d, f, h, i, k\}$ are identified as I_{DDQ} equivalent and therefore faults in $\{\langle\{a, b, c, d, f, h, i, k\}, \{v\}\rangle\}$ are identified as I_{DDQ} equivalent.□

To identify first type of equivalent faults, for each pair of equivalent fault sets, we search the fault lists to see if such a structure (as shown in Figure 4 (a)) exists. If there is such a structure, the pair of equivalent fault sets are equivalent and can be merged into one. Continuing the example in Figure 5 faults $\langle\{e, g, j\}, \{u\}\rangle$ are three equivalent faults. From structure analysis, we know faults $\langle e, u, \rangle$ and $\langle a, v\rangle$ are equivalent. Therefore, we conclude that faults in $\{\langle\{a, b, c, d, f, h, i, k\}, \{v\}\rangle\}$ and $\{\langle\{e, g, j\}, \{u\}\rangle\}$ are I_{DDQ} equivalent.

Although these two types of equivalences are simple, they represent a large portion of equivalent faults and can greatly reduce the number of faults needed to be processed.

3.2.4 Deterministic Test Generation

After collapsing I_{DDQ} equivalent faults, faults related in a "simple way" are identified. Faults remaining in fault list now are related in a more "complicated way". For pairs of such faults, a deterministic procedure similar to redundant identifying stuck-at faults is used to identify whether they are equivalent or not. Therefore, **in the third phase, test vectors which distinguish all pairs of as yet undistinguished faults are deterministically generated to complete the diagnosis process**.

To show two faults $f = \langle x, y\rangle, g = \langle u, v\rangle$ are equivalent, we need to show that for all input combinations $V(x) \neq V(y)$ iff $V(u) \neq V(v)$. This is equivalent to showing that the stuck-at 0 fault in Figure 6 at line z is a redundant fault. Since if f and g are equivalent we can never set line z to 1. Otherwise the test generated for detecting z stuck-at-0 fault can distinguish f, g.

In the last phase for all pairs of faults in fault list, by adding the circuitry of Figure 6, a new circuit is constructed and handed over to a deterministic stuck-at test generator. If a test vector can be generated then the two faults are not identical and the test vector is used for further diagnosis. Otherwise the two faults are identical and one of them will be removed from fault list. Such a process continues until we exhaust all pairs of faults.

A straight forward implementation of above approach for phase 3 will result in a larger test set — for each distinguishable pair of faults, we will have a test vector. This is not necessary. Some faults are equivalent. Test which distinguishes one pair of faults may also distinguish other pairs of faults.

In order to reduce the size of test set, we further divide phase 3 into two sub-phases. In first sub-phase, we use heuristic to select fault pairs which are most likely to be distinguished. Then tests for distinguish these pairs of faults are generated and applied to the circuit to reduce the number of faults in fault list. After that in the second sub-phase, we generates test for distinguishing all fault pairs in fault list.

The heuristic we used in first sub-phase is to select pairs of faults with different 0(1)-controllability. Although this is not always true, but faults with different 0(1)-controllability are more likely to be "non-equivalent".

After applying the test vectors generated in phase 3, the remaining faults, with their equivalences, are the possible set of faults. These faults are I_{DDQ} equivalent and can not be distinguished further by I_{DDQ} measurements alone. Hence the diagnosis is complete.

4 Experimental Results

In this section, we present our experimental results. We compare our results with results from other systems. The proposed system was implemented using C++ on SUN 4. ISCAS85 and scan version of ISCAS89 benchmark circuits were used in our experiment. In the experiment we randomly generated 500 faults. These faults are used to simulate 500 faulty circuits.

The experimental results were obtained using METAL fault model. METAL contains all BFs involving two gate outputs only. Any other set of faults can also be used. We used this extensive fault model to stretch the system.

We present the performance of our system and compare it with a static diagnostic test generation system(SDTG)[8], as well as, a system that uses stuck-at test sets. The rationale for comparing with a stuck-at test set is twofold. Firstly, it might be possible to use the stuck-at test set, which is readily available, for diagnosis. In this case, we are interested in ascertaining if this approach is superior to what we are proposing. Secondly, one could start with the available stuck-at test set (rather than the random test set we use) and augment it in a similar manner. In this case we are interested in knowing if this leads to a better diagnosis approach.

We call the remaining faults after diagnosis **residual faults**. In our comparison we use the following figures of merit. The most important criteria is the number of residual faults. In our approach the residual faults always reduces to one equivalence class. The second important criteria is the size of the test set. Size of the test set is very important for I_{DDQ} measurement based analysis because I_{DDQ} measurement is a very slow process.

Table 1 presents the performance of our DDTG algorithm for the 500 random faults. The data under column *Residual Faults* are the (average *Ave*, maximal *Max*) number of residual faults over those 500 instances. Note that the number of faults cannot be reduced any further. They are the best you can get if you use only I_{DDQ} measurement for diagnosis. Data under column *Vectors*(*Time*) are the (average, maximal) number of input vectors generated(time used, in CPU seconds) during the dynamic diagnostic test generation process. Note that Time does not include the time for down loading the vectors into the tester, making the I_{DDQ} measurements and getting back the faulty response.

Table 2 presents a comparison between our system and the system which uses stuck-at test sets for diagnosis. The stuck-at test set used is from ATALANTA[20]. The data under DDTG is from our system, and the data under SSF is from using the stuck-at test set. Time for SSF is only the time required for fault dropping. The two systems share all the diagnosis procedures except that our system generates the tests dynamically and SSF uses the precomputed stuck-at test set.

For small circuits, the number of residual faults are about the same in both cases. But for large circuits this is not true. The DDTG system gives substantially better results, both on an average as well as in the worst case. For example, see S15850.1, S35932, S38584.1, $\cdots$. Note that DDTG will never result in a larger residual set than SSF because DDTG is a complete diagnosis system where as SSF is not. In addition, we note that the performance of SSF, so far as reducing the number of faults is concerned, is inconsistent.

Another important advantage of DDTG over SSF is the small size of the test set. In all cases (except for C6288), the average number of I_{DDQ} measurements required, if DDTG is used, is substantially smaller. The average is almost always less than half the SSF test set size. Even if we take the worst case number of vectors generated by DDTG it is substantially smaller than the stuck-at test set. This is very crucial for I_{DDQ} measurement based diagnosis since I_{DDQ} measurement is very time consuming and can adversely affect the total diagnosis time. In addition, it also points to the fact that a stuck-at test set is not a good starting point for dynamic diagnostic test generation.

Table 3 gives the comparison between our system and SDTG system[8]. The data under DDTG are the average and maximal number of vectors(times) over 500 instances. The data under SDTG are the vectors(times) the system generated(used). We have not reported data on the residual set size because DDTG will always out perform SDTG. On an average DDTG requires much less measurements (smaller test set). However, the major difference between these two systems, as we can see from the given table, is the test generation time. Even the worst case time for DDTG is order of magnitude smaller than SDTG. In fact, SDTG could not be used to generate tests for any thing beyond C7552. On the other hand we were able to use DDTG effectively for all the benchmark circuits.

References

[1] M. Abramovici and M. A. Breuer. “Multiple Fault Diagnosis of MOS Combinational Networks”. *IEEE Trans. on Computers*, C-29(6):451–460, June 1980.

[2] M. Abramovici and M. A. Breuer. “Fault Diagnosis in Synchronous Sequential Circuits Based on an Effect-Cause Analysis”. *IEEE Trans. on Computers*, C-31(12):1165–1172, Dec 1982.

[3] M. Abramovici, M. A. Breuer, and A. D. Friedman. *“DIGITAL SYSTEMS TESTING AND TESTABLE DESIGN”*. Computer Science Press, 1990.

[4] R. C. Aitken. “Fault Location with Current Monitoring”. In *IEEE International Test Conference*, pages 623–632, 1991.

[5] V. Boppana and W. K. Fuchs. “Fault Dictionary Compaction by Output Sequence Removal”. In *Proceedings of International Conference on Computer-Aided Design*, 1994.

[6] D. J. Burns. “Locating High Resistance Shorts in CMOS Circuits by Analyzing Supply Current Measurement Vectors”. *Int’l Symposium for Testing and Failure Analysis*, pages 231–237, 1989.

[7] P. Camurati, D. Medine, P. Prinetto, and M. S. Reorda. “A diagnostic test pattern generation algorithm”. In *Proceedings of the International Test Conference*, pages 52–58, 1990.

[8] S. Chakravarty, K. Fuchs, and J. Patel. “Evaluation and Generation of I_{DDQ} Diagnostic Tests for Bridging Faults in Combinational Circuits”. Technical Report 95-11, State University of New York at Buffalo, Dept. of Computer Science, 1995.

[9] S. Chakravarty and Y. Gong. “An Algorithm for Diagnosing Two-Line Bridging Faults in Combinational Circuits”. In *Design Automation Conference*, pages 520–524, 1993.

[10] S. Chakravarty and Y. Gong. “Diagnostic Simulation of Stuck-at Faults in Combinational Circuits”. In *IEEE VLSI Test Symposium*, pages 128–133, 1994.

[11] S. Chakravarty, Y. Gong, and S. Venkataraman. “Diagnostic Simulation of Stuck-at Faults in Combinational Circuits”. Submitted to Journal of Electronic Testing: Theory and Applications.

[12] S. Chakravarty and M. Liu. “I_{DDQ} Measurement Based Diagnosis of Bridging Faults”. In *Journal of Electronic Testing: Theory and Application (Special Issue on I_{DDQ} Testing)*. Kluwer Academic Publishers, 1993.

[13] S. Chakravarty and S. Suresh. "I_{DDQ} Measurement Based Diagnosis of Bridging Faults in Full Scan Circuits". In *7th International Conference on VLSI Design*, pages 179–182, 1994.

[14] F. J. Ferguson and T. Larrabee. "Test Pattern Generation for Realistic Bridging Faults in CMOS ICs". In *Proceedings of the International Test Conference*, pages 492–499, 1991.

[15] J. Ferguson and J. P. Shen. "Extraction and Simulation of Realistic CMOS Faults using Inductive Fault Analysis". In *Proceedings of the International Test Conference*, pages 475–484, 1988.

[16] M. R. Garey and D. S. Johnson. *"Computers and Intractability — A Guide to the Theory of NP-Completeness"*. W. H. Freeman and Company, 1979.

[17] T. Grüning, H. Koopmeiners, and U. Mahlstedt. "DIAGTEST: A Fast Diagnostic Test Pattern Generator For Combinational Circuits". In *Digest of Papers, International Conference on Computer-Aided Design*, pages 194–197, 1991.

[18] S. K. Jain and V. D. Agrawal. "STAFAN: An Alternative to Fault Simulation". In *Design Automation Conference*, pages 18–23, 1984.

[19] K. Kubiak, S. Parkes, W. K. Fuchs, and R. Saleh. "Exact Evaluation of Diagnostic Test resolution". In *Proceedings 29th Design Automation Conference*, pages 347–352, 1992.

[20] H. K. Lee and D. S. Ha. "On the Generation of Test Patterns for Combinational Circuits". Technical Report 12-93, Dept. of Electrical Eng., 1993.

[21] T. Lee, W. Chuang, I. N. Hajj, and W. K. Fuchs. "Circuit-Level Dictionaries of CMOS Bridging Faults". In *VLSI Test Symposium*, pages 386–391, 1994.

[22] M. Melton and F. Brglez. "Automatic Pattern Generation for Diagnosis of Wiring Interconnect Faults". In *Proceedings of the International Test Conference*, pages 389–398, 1992.

[23] I. Pomeranz and S. M. Reddy. "On the Generation of Small Dictionaries for Fault Location". In *Proceedings of International Conference on Computer-Aided Design*, pages 272–279, 1992.

[24] I. M. Ratiu, A. Sangiovanni-Vincentelli, and D. O. Pederson. "VICTOR: A Fast VLSI Testability Analysis Program". In *IEEE Test Conference*, pages 397–401, 1982.

[25] R. S. Reddy, I. Pomeranz, S. M. Reddy, and S. Kajihara. "Compact Test Generation for Bridging Faults under I_{DDQ} Testing". To appear at VLSI Test Symposium 1995.

[26] E. M. Rudnick, W. K. Fuchs, and J. H. Patel. "Diagnostic Fault Simulation of Sequential Circuits". In *Proceedings of the International Test Conference*, pages 178–186, 1992.

[27] P. G. Ryan, W. K. Fuchs, and I. Pomeranz. “Fault Dictionary Compression and Equivalence Class Computation for Sequential Circuits”. In *Proceedings of International Conference on Computer-Aided Design*, pages 508–511, 1993.

[28] P. G. Ryan, S. Rawat, and W. K. Fuchs. “Two-Stage Fault Location”. In *Proceedings of the International Test Conference*, pages 963–968, 1991.

[29] R. E. Tulloss. “Size Optimization of Fault Dictionaries”. In *Digest of Papers 1978 Semiconductor Test Symp.*, pages 264–265, Oct 1978.

[30] R. E. Tulloss. “Fault Dictionary Compression: Recognizing When a Fault May Be Unambiguously Represented by a Single Failure Detection”. In *Digest of Papers 1980 Test Conf.*, pages 368–370, Nov 1980.

[31] S. Venkatraman, I. Hartanto, W. K. Fuchs, E. M. Rudnick, S. Chakravarty, and J. H. Patel. “Rapid Diagnostic Fault Simulation of Stuck-at Faults in Sequential Circuits Using Compact lists”. To appear in 1995 Design Automation Conference.

Circuit	Residual Faults		Vectors		Time	
	Ave	Max	Ave	Max	Ave	Max
C432	1.398	40	19.66	34	0.2977	1.43
C499	3.058	80	30.07	48	0.3819	0.83
C880	2.11	85	24.36	48	0.5377	3.89
C1355	2.186	112	40.83	83	0.7804	2.8
C1908	28.93	1131	36.62	69	3.691	83.18
C2670	9.9	1024	27.33	54	8.095	124.7
C3540	13.73	265	37.84	106	4.248	171.2
C5315	13.66	1746	29.98	61	68.68	29950
C6288	1.312	17	28.05	47	3.392	125.3
C7552	8.93	197	33.24	68	7.365	87.55
S208.1	2.746	44	22.46	35	0.3747	1.33
S298	3.156	59	19.33	26	0.2593	1.09
S382	3.02	78	21.33	29	0.3613	3.02
S386	1.738	8	23.09	38	0.4224	1.39
S526n	3.066	82	21.19	32	0.3614	3.32
S641	17.34	504	24.54	39	0.7708	64.2
S713	15.47	138	25.17	42	0.7293	31.62
S820	2.08	60	25.93	56	0.4928	5.05
S838	1.76	5	60.64	81	1.194	2.47
S953	1.194	3	44.79	67	0.9994	2.15
S1196	2.26	173	33.51	69	0.9722	7.3
S1238	1.194	4	32.56	68	0.8032	2.76
S1423	8.84	445	26.91	52	1.029	14.51
S5378	22.67	3755	48.19	140	15.82	5583
S13207.1	102.2	21384	72.16	250	141.6	55650
S15850.1	53.8	2595	59.07	259	52.39	9846
S35932	134.6	2729	35.91	49	20.63	64.05
S38417	63.4	2565	59	306	58.26	854.6
S38584.1	3257	259364	95.74	436	855.1	211900

Table 1: Performance of DDTG for METAL

Circuit	Residual Faults				Vectors				Time			
	Ave		Max		Ave		Max		Ave		Max	
	DDTG	SSF	DDTG	SSF	DDTG	SSF	DDTG	SSF	DDTG	SSF	DDTG	SSF
C432	1.398	1.398	40	40	19.66	47	34	47	0.2977	0.186	1.43	0.44
C499	3.058	3.058	80	80	30.07	53	48	53	0.3819	0.2245	0.83	0.37
C880	2.11	2.166	85	89	24.36	53	48	53	0.5377	0.4053	3.89	0.85
C1355	2.186	2.186	112	112	40.83	85	83	85	0.7804	0.7356	2.8	1.3
C1908	28.93	28.93	1131	1131	36.62	120	69	120	3.691	1.452	83.18	4.75
C2670	9.9	9.9	1024	1024	27.33	105	54	105	8.095	2.614	124.7	6.41
C3540	13.73	13.73	265	265	37.84	153	106	153	4.248	3.362	171.2	4.12
C5315	13.66	13.66	1746	1746	29.98	113	61	113	68.68	4.137	29950	13.67
C6288	1.312	1.836	17	68	28.05	28	47	28	3.392	1.329	125.3	1.64
C7552	8.93	8.93	197	197	33.24	210	68	210	7.365	10.46	87.55	12.13
S208.1	2.746	2.8	44	44	22.46	34	35	34	0.3747	0.08334	1.33	0.17
S298	3.156	3.156	59	59	19.33	33	26	33	0.2593	0.08536	1.09	0.2
S382	3.02	3.116	78	78	21.33	36	29	36	0.3613	0.125	3.02	0.36
S386	1.738	1.764	8	19	23.09	71	38	71	0.4224	0.1755	1.39	0.31
S526n	3.066	3.066	82	82	21.19	59	32	59	0.3614	0.2054	3.32	0.48
S641	17.34	17.34	504	504	24.54	59	39	59	0.7708	0.3947	64.2	1.39
S713	15.47	17.3	138	443	25.17	55	42	55	0.7293	0.3776	31.62	0.98
S820	2.08	2.08	60	60	25.93	114	56	114	0.4928	0.485	5.05	1.08
S838	1.76	1.828	5	5	60.64	101	81	101	1.194	0.7363	2.47	1.67
S953	1.194	1.23	3	3	44.79	86	67	86	0.9994	0.561	2.15	1.16
S1196	2.26	2.26	173	173	33.51	144	69	144	0.9722	1.005	7.3	2.98
S1238	1.194	1.218	4	4	32.56	149	68	149	0.8032	1.008	2.76	1.26
S1423	8.84	13.91	445	880	26.91	71	52	71	1.029	0.8257	14.51	1.55
S5378	22.67	22.77	3755	3756	48.19	251	140	251	15.82	9.695	5583	46.29
S13207.1	102.2	103.2	21384	21567	72.16	469	250	469	141.6	51.64	55650	366.5
S15850.1	53.8	114.2	2595	32765	59.07	440	259	440	52.39	54.15	9846	164.9
S35932	134.6	10266	2729	446670	35.91	70	49	70	20.63	20.97	64.05	35.63
S38417	63.4	63.4	2565	2565	59	888	306	888	58.26	269.1	854.6	302.5
S38584.1	3257	5226	259364	1010106	95.74	676	436	676	855.1	475.8	211900	49400

Table 2: Comparison with SSF Test Set (METAL)

Circuit	Vectors			Time		
	DDTG		SDTG	DDTG		SDTG
	Ave	Max		Ave	Max	
C432	19.66	34	27	0.2977	1.43	99.50
C499	30.07	48	40	0.3819	0.83	140.42
C880	24.36	48	37	0.5377	3.89	287.07
C1355	40.83	83	78	0.7804	2.8	668.43
C1908	36.62	69	74	3.691	83.18	1116.73
C2670	27.33	54	36	8.095	124.7	1331.03
C3540	37.84	106	65	4.248	171.2	2246.38
C5315	29.98	61	45	68.68	29950	3415.43
C6288	28.05	47	53	3.392	125.3	3104.72
C7552	33.24	68	58	7.365	87.55	8230.52
S208.1	22.46	35	33	0.3747	1.33	81.03
S298	19.33	26	21	0.2593	1.09	61.05
S382	21.33	29	23	0.3613	3.02	95.25
S386	23.09	38	29	0.4224	1.39	99.80
S526n	21.19	32	29	0.3614	3.32	129.58
S641	24.54	39	36	0.7708	64.2	273.40
S713	25.17	42	38	0.7293	31.62	298.13
S820	25.93	56	43	0.4928	5.05	279.47
S953	60.64	67	68	1.194	2.15	581.63
S1196	44.79	69	55	0.9994	7.3	626.58
S1238	33.51	68	70	0.9722	2.76	703.80
S1423	32.56	52	40	0.8032	14.51	687.45
S5378	26.91	140	113	1.029	5583	10152.9

Table 3: Comparison with SDTG System (METAL)

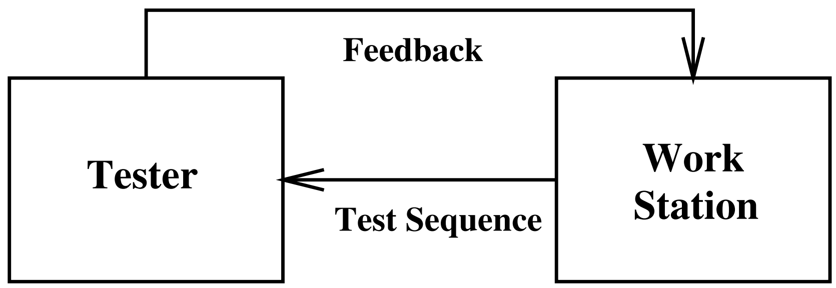

Figure 1: Diagnosis System

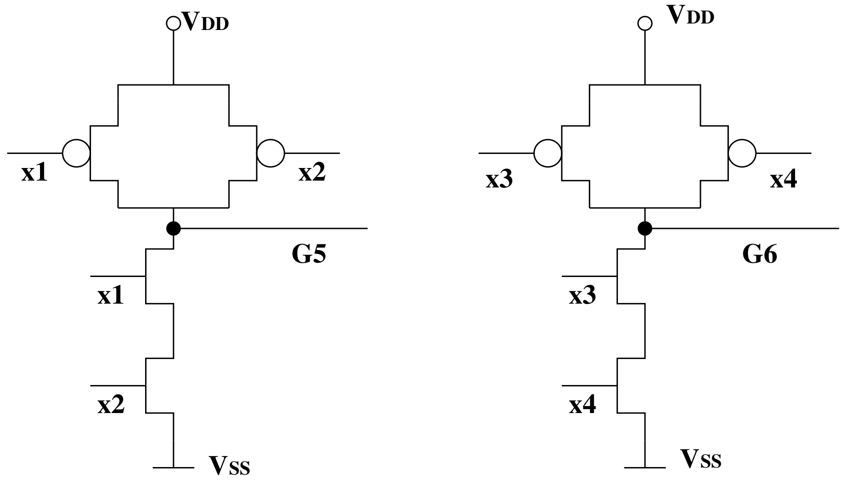

Figure 2: Example Circuit

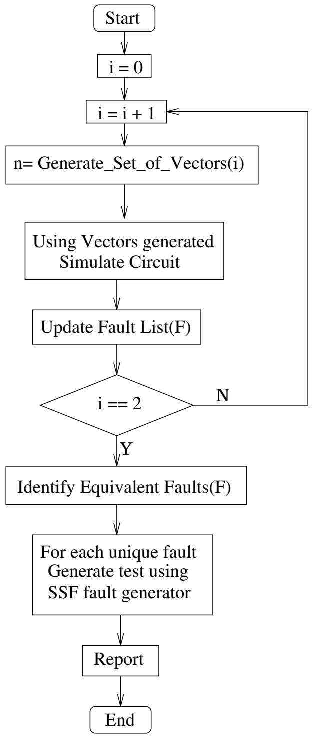

Figure 3: Dynamic Diagnostic Test Generation

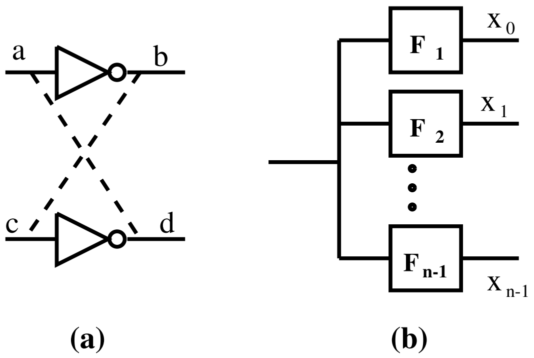

Figure 4: Simple Equivalent Faults

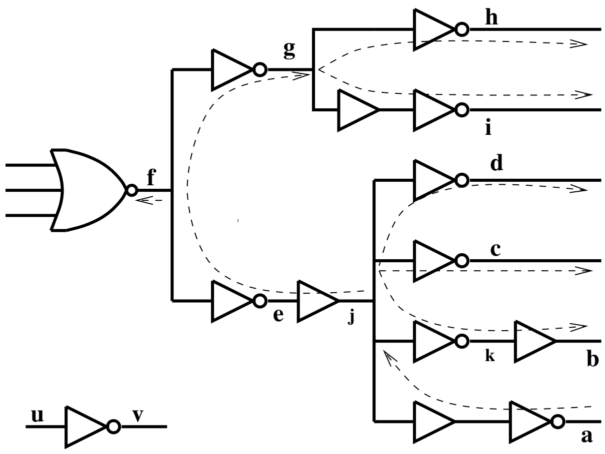

Figure 5: Identify Equivalent Faults

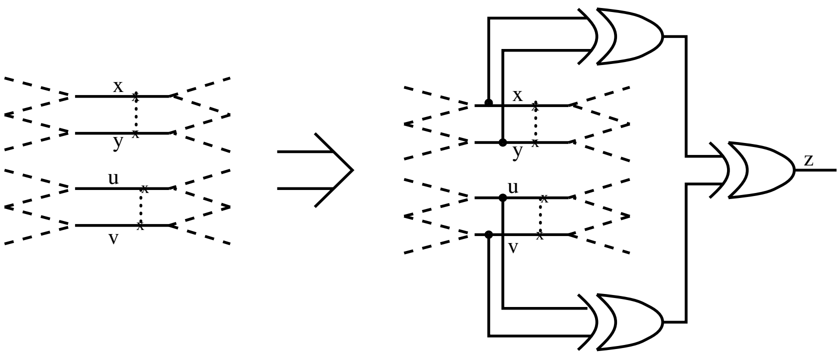

Figure 6: Construct New Circuit for SSA TG