Signature Based Diagnosis for Logic BIST

Wu-Tung Cheng, Manish Sharma, Thomas Rinderknecht, Liyang Lai and Chris Hill

Mentor Graphics Corporation
8005 SW Boeckman Road
Wilsonville, OR 97070, USA

Abstract

This paper presents a new approach for performing logic BIST Diagnosis exclusively using MISR signatures. Unlike conventional logic BIST diagnosis approaches which require either huge test time or complicated logic BIST design and ATE flow, signature based diagnosis does not require dynamically changing MISR operations for each failing device. Our experimental data shows that signature based diagnosis can achieve similar diagnosis resolution with manageable diagnosis run time while eliminating most of the complexity associated with the traditional approach to logic BIST diagnostics.

1. Introduction

In VLSI circuit manufacturing, scan-based testing and fault diagnosis play an important role. Testing screens out chips with defects and then diagnosing those defects helps to determine the exact location and probable cause of the failure. The ability to diagnose failures directly from the logic BIST patterns is important for several reasons. It is necessary to identify failures that occur when logic BIST fails during system test. It is also useful to identify failures that occur when logic BIST is used as part of the manufacturing test sequence. Since logic BIST is based on the application of a large number of pseudo-random patterns, there is always some chance that a failure will be detected during the logic BIST test that is not activated using conventional ATPG vectors. In this case, the ability to perform diagnostics directly from the logic BIST pattern set is essential to identifying the cause of the failure.

Advances in scan diagnostics have led to tools capable of identifying the defect types, such as bridges or opens, as well as the potential defect location on the die [1]. These tools are used to post process the tester fail information that is captured after applying ATPG vectors. For ATPG vectors, the tester fail information can be directly mapped to failing scan cells. For logic BIST, single MISR signature is sufficient to identify failing devices [2][3]. However, the signature does not have enough information to identify the locations of detected defects [4]. In order to do diagnosis, some chose to unload the exact scan cell information in diagnostic mode and perform the same diagnosis as regular scan ATPG [5]. But this method is expensive since it requires special ATE interface. Others chose to use multiple signatures obtained by applying the same logic BIST pattern set many times with different configurations [6][7][8][9]. These configurations can be obtained either by reconfiguring programmable MISR in different sessions [6] or by scan cell masking/partitioning [7][8][9]. Normally a large number of logic BIST reruns are required and it incurs excessive test application time in diagnostic mode. Cycling registers [10][11] are also used to help identify error bit information but the probability of diagnostic aliasing is pretty high. The authors in [19] propose to observe a compacted MISR signature in every test cycle for each pattern. This information can be used to try to recover the failing flops. However, this method requires significant change in the compactor structure and test data volume overhead.

Traditional logic BIST diagnostics scheme are based on the idea of unloading the exact scan cell information. It relies on the signature to identify failing patterns, but then reverts to direct scan chain access to unload the failing scan cell data. This three-phase flow is described below:

1. Use the MISR to identify failing device (unload single MISR value at end of test)

2. Use the MISR to identify failing patterns (unload and reset MISR at end of each pattern, or perform binary search)

3. Bypass the MISR to identify failing scan cells (To do this phase effectively, only failing patterns are reapplied at bypass mode to unload failing scan cells. Since failing patterns are not the same for all Devices under Test (DUT). Therefore, DUT specific test vectors are needed to unload failing scan chain contents)

The uses of DUT specific test vectors in Phase 3, and a binary search algorithm in Phase 2, require a tight link between the tester, and the software that generates these dynamic vectors. Traditional testers are designed for use with ATPG vectors, and work well in batch oriented flows. Automated support for dynamically generated test

1-4244-0292-1/06/$20.00 © 2006 IEEE

vectors requires modifications to the tester and the development of the corresponding logic BIST software.

In contrast to traditional logic BIST diagnostics flow, our new methodology, *signature based diagnosis*, identifies potential fault candidates directly from the MISR signatures available in Phase 2. This eliminates the need for dynamic vector generation, and provides a diagnostics flow that will work well with batch oriented testers without requiring a special tester interface.

Recently, a new algorithm to diagnose compressed failing data was proposed [12] and used successfully for circuits with space compactors [13]. We extend this new algorithm to MISR signatures. Here, we agree that one signature from all logic BIST patterns is not sufficient for diagnosis. However, if we can have one signature per pattern, the information may be sufficient to do diagnosis. This approach is called *signature based diagnosis* here. Using signature based diagnosis, we do not have to bypass the MISR to access all scan cell data which makes it possible to acquire all the necessary failure data using a single set of pre-computed test vectors.

To support signature based diagnosis, there are two phases in our logic BIST test flow. In Phase 1, our logic BIST operation is same as others. All test responses are compacted into one MISR signature. At the end of this phase, based on this final signature, we decide whether the device is defective or not. For a non defective device, test is complete. If this is a defective device, Phase 2 is activated.

In Phase 2, logic BIST operation is similar to Phase 1, except it will unload the MISR signature after each pattern is finished. To ensure that the MISR signature of each pattern is independent of other failing patterns, we reset the MISR signature at the beginning of each pattern. Instead of resetting to constant values, during signature unloading, we load the MISR with the expected good machine signature. This way we ensure both phases have the same good circuit simulation to save simulation time and thus avoid the problem of salvaging test windows as in [14]. The impact of Phase 2 on total tester runtime is minimal, since it is only applied to failing devices which are identified during Phase 1.

The rest of this paper is organized as follows. In Section 2 we describe the enhancements needed in logic BIST controllers to enable per-pattern MISR signature collection in a single run even using slow-speed ATEs. In Section 3, we describe signature based diagnosis algorithm based on these collected per-pattern MISR signatures. In Section 4, two industrial circuits are used to validate signature based diagnosis, followed by the conclusions in Section 5.

2. Logic BIST Controller Architecture

Our logic BIST controller follows the STUMPS [15] architecture and has a state machine that controls the logic BIST session. Table 1 describes the basic components in our logic BIST controller:

Table 1: Logic BIST Controller

PRPG	Pseudo-random pattern generator Load pattern data into scan chains
MISR	Multiple Input Signature Register Generate signature based data unloaded from scan chains
Pattern counter	Track total number of patterns that have been applied
Shift counter	Track number of shift cycles applied for this pattern
Controller	State machine controls sequencing of test – transitions from shift to capture, etc.
Clock generator	Generate internal clock signal used during shift operations, and also high speed clock pulses during capture
Interface logic	Control registers define parameters of current run – includes initial PROG value, number of patterns, etc.

In order to support at-speed operation, our logic BIST controller is driven by a single high-speed clock. This high speed clock is used to generate a slower internal clock that drives all of the logic (PRPG, MISR, etc.) during shifting. During the capture window, we inject high speed clock pulses into the core using the clock generator circuit.

The logic BIST hardware includes several configuration registers that are used to define parameters such as the initial PRPG and MISR values, and the total number of patterns to apply. One common scenario is to control the logic BIST test via an 1149.1 compliant interface. This means that our interfacing registers (including the current MISR value) must be accessible via an asynchronous interface that is driven from the clock signal named TCK which is used by the 1149.1 interface. The control registers in the interface block are driven using TCK. Once a new value has been loaded into these registers, a signal is sent to the state machine. This signal triggers the transfer of the stable value from the registers in the interface block

(driven by TCK), into the appropriate internal registers driven by the free running clock, as shown in Figure 1.

The internal sequencing of operations within the logic BIST controller is driven from the state machine. In order

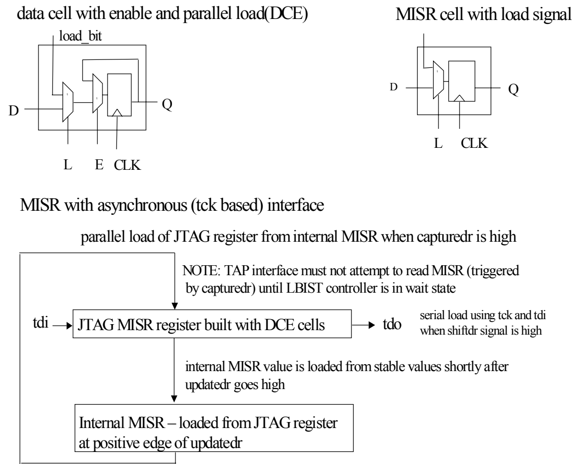


Figure 1: Synchronization Logic

to provide a mechanism for unloading and re-loading the MISR at the end of each pattern, we have modified the state machine so that it will go into a wait state after performing the capture operation. The interface logic (typically driven from TCK) will send a signal to the state machine once the current MISR value has been unloaded, and the "expected" MISR value has been re-loaded. Given an architecture that already supports the use of an asynchronous interface for accessing internal configuration registers, the additional logic that is required to support diagnostics mode is minimal.

3. Diagnostics Algorithm

The *signature based diagnosis* algorithm identifies possible failing locations from a failing MISR signature for each failing pattern without requiring any additional information. This algorithm is based on a new compactor independent direct diagnosis [12] technique. It takes in as input failing MISR signatures and the corresponding failing patterns and produces suspect failing locations that best describe the failing signatures. This is done in two main steps:

1) Determine an *error function* that describes how errors in the scan cell unload values affect the MISR signature.
2) Use this error function to perform compactor independent direct diagnosis.

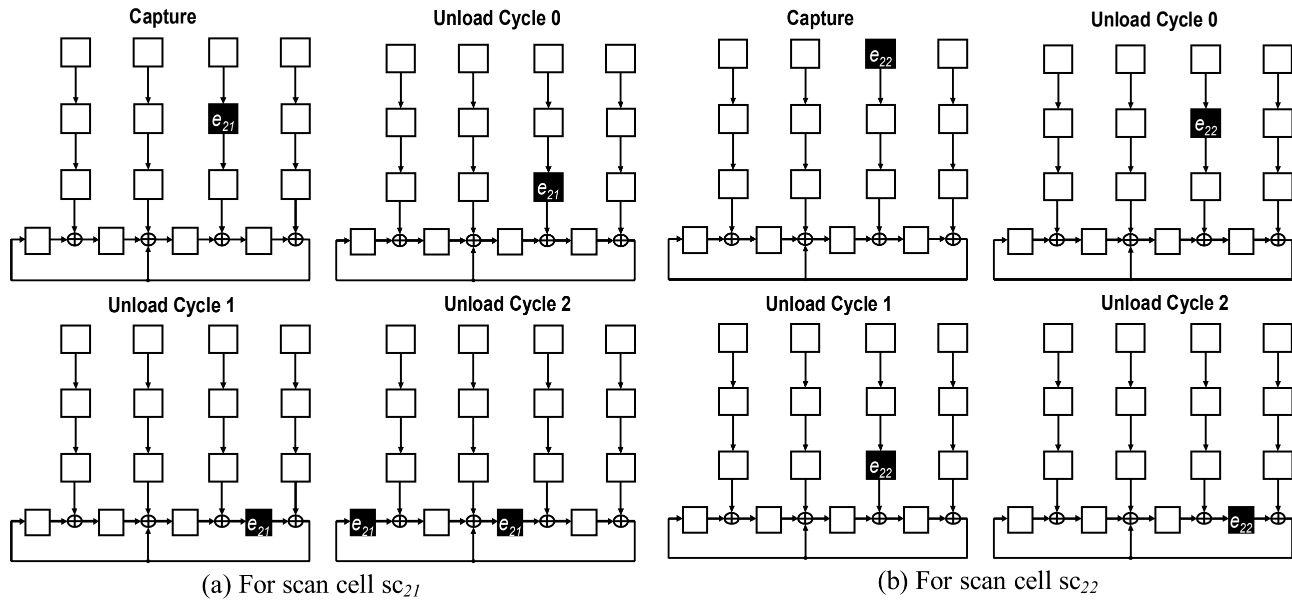
The next three sub-sections describe these two steps in more detail. We first describe the meaning of the error function. Then, we describe how to determine this error function for a MISR compactor. Finally, we present how the error function is used in diagnosis. In all the three subsections we use an example circuit in which the number of scan chains is equal to the MISR size. This is just done for the sake of simplifying the presentation. The techniques described here easily extend to other logic BIST configurations for example where an XOR-tree space compactor is present between the scan chains and the MISR.

The Error Functions

As mentioned before the error function describes how failing scan cell unload values affect the MISR signature. Initially, let us consider the case of a logic BIST controller with a MISR connected directly to a number of scan chains, one chain per MISR bit. We shall defer until later the analysis involved for more complex logic BIST controllers.

In order to describe the error function formally we first introduce some terminology. Let sc_{ij} denote a scan cell in the design where i denotes the scan chain number ($i = 0$ being the scan chain connected to the leftmost MISR bit), and j denotes the scan cell number within the chain ($j = 0$ being the scan cell closest to scan out). Let e_{ij} be a Boolean variable which is equal to **1** if there is an error in the scan unload value in scan cell sc_{ij} (i.e. the unload value for scan cell sc_{ij} is not what is expected) for some failing test pattern. Let $\mathbf{e}$ be the vector of all e_{ij} variables, i.e. $\mathbf{e} = [e_{00}, e_{01}, \cdots]$. In other words the vector $\mathbf{e}$ describes the positions of all the scan cells which have erroneous values for some failing test pattern.

Now the error function can be defined for each MISR bit as a Boolean valued function in $\mathbf{e}$. For the *k*-th MISR bit this function is denoted by $\varepsilon_k(\mathbf{e})$. The error function is **1** for all those (and only those) values of $\mathbf{e}$ such that if the corresponding failing test pattern is unloaded into the MISR, the *k*-th MISR bit in the resulting signature will have an erroneous value. Note that since the MISR is a linear device, the location of failing MISR bits in the MISR signature corresponding to a failing test pattern



(a) For scan cell sc_{21} (b) For scan cell sc_{22}

Figure 2: **Symbolic Representation of MISR Emulation**

does not depend on the actual scan unload values. It only depends on which scan cells contain erroneous values for the failing test pattern. This is the fact that allows us to define the error function in the above manner.

Determining the Error Function

In order to determine the error function for each MISR bit we first determine how an error in a single scan cell will affect the MISR signature, or in other words the *error signature* of a single scan cell error. Next, based on the linearity of the MISR compactor, all such signatures can be added for each MISR bit to obtain the corresponding error function for each bit. The effect of an error in a single scan cell, say sc_{ij}, can be determined by emulating the process of unloading the scan cell values into the MISR. In order to do this the unload values for all scan cells, except sc_{ij}, are set to **0**. The unload value for cell sc_{ij} is set to the symbol e_{ij}. The MISR initial values are also all set to **0**. The MISR signature corresponding to the above unload values is then determined by emulating the process of unloading these values into the MISR. This process is illustrated for an example logic BIST compactor configuration in Figure 2. This circuit has four scan chains with three cells each unloading into a four bit MISR. The figure illustrates the emulation process for an error in sc_{21} and sc_{22}. In a similar manner the error signatures for all the scan cells in the design are obtained. These error signatures are then added at all the MISR bits to obtain the error function for each bit. For our example circuit this is illustrated in Figure 3. In this figure, each horizontal row of light squares beneath the MISR is an error signature of some scan cell.

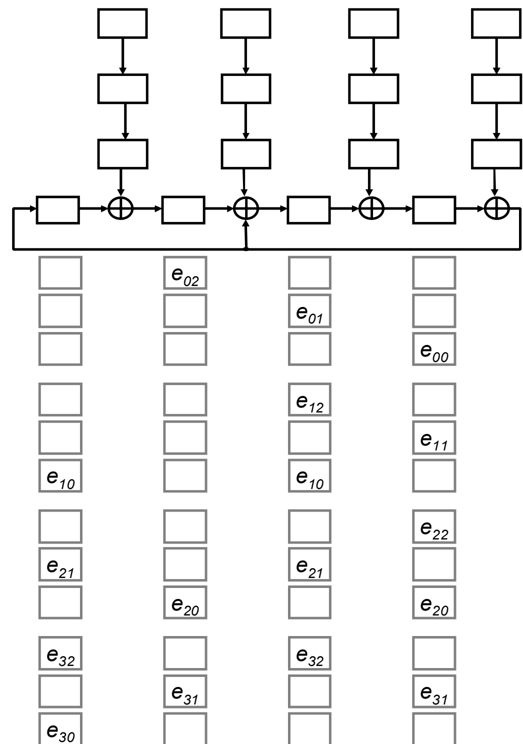


Figure 3: Error Signatures for all Scan Cells

Hence in Figure 3 the Boolean sum of each vertical column gives the error function for the corresponding MISR bit. The reader should note that, in this description, symbolic simulation is used just for the sake of clarity. In

actual implementation symbolic simulation is not required and is not used.

A brute force implementation of the method described above would result in the emulation of $n \cdot l$ scan shift cycles, where n is the total number of scan cells in the design and l is the number of shift cycles in one unload. However, a simple observation allows us to make the process much more efficient by avoiding repetitive work. It can be seen in Figure 2 that the error signature for scan cell sc_{22} is exactly the same as the error signature obtained at the end of the second shift cycle while the signature for scan cell sc_{21} is being computed. In other words the error signature for sc_{22} is also determined while we are computing the error signature for sc_{21}. In general the error signature for $sc_{i(j+x)}$ is determined within the process of computing the error signature for sc_{ij} because the former is just structurally shifted x time steps behind the latter. So in order to compute the error signatures for all the scan cells we just need to run the full emulation for the zero-th scan cell, sc_{i0} in each scan chain. This reduces the number of shift cycles that need to be emulated to $m \cdot l$ where m is the number of scan chains in the design.

At this point the reader should note that the technique described above extends to alternate logic BIST compactor configurations, e.g. where there is a XOR-tree space compactor preceding the MISR etc. Such additional logic, between the scan chains and the MISR, is linear in nature and hence the error function computation can be extended to operate through it.

Using the Error Function in Diagnosis

So far we have described what the error function is and how it can be efficiently determined. In this subsection we present how this error function can be used for diagnosis. At a high level, logic diagnosis follows the following steps:

Step 1 For each failing pattern find failing locations that explain the observed failure based on the single location at a time paradigm [16].

Step 2 Analyze the suspect fail locations found in the previous step to determine the most likely candidates as well as check fault activation conditions to further identify fail causes as bridges, opens etc[17].

The first step in the diagnosis process can also be divided into two main steps which are performed for each failing pattern:

Step 1.1 Determine a list of initial candidate fail locations, through critical-path based back cone tracing [18], which can potentially explain the failing pattern.

Step 1.2 Perform fault simulation on the initial candidates found above for the current failing pattern to exactly determine which faults locations explain the observed failing behaviour for the current pattern.

Now for a circuit without any compaction the fail information tells us which scan cells are failing for each failing pattern so that performing back cone tracing in Step 1.1 above is straightforward. However, for the case of logic

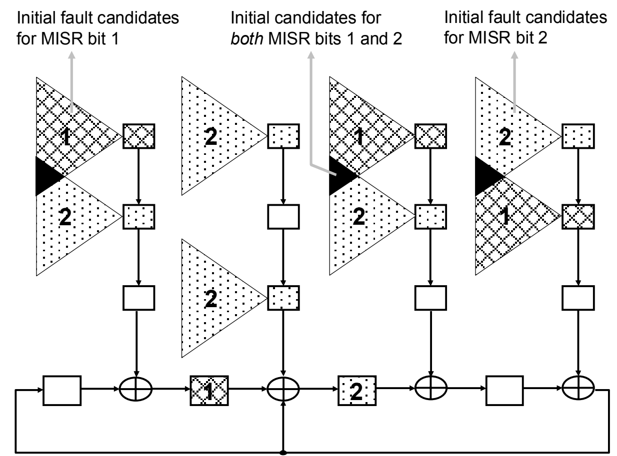


Figure 4: Determining initial candidates fail locations from MISR fail signature

BIST we only know the failing MISR signature for each failing pattern, so we cannot directly perform the back trace. This is where the error function is used. This is illustrated using the following example. Consider again the circuit in Figure 3. Let us say that for some failing pattern, failures are observed in MISR bits one and two. Using our previous analysis of error functions for this circuit configuration, the error functions for MISR bit one and two are:

$$\varepsilon_1 = e_{02} + e_{20} + e_{31}$$

$$\varepsilon_2 = e_{01} + e_{12} + e_{10} + e_{21} + e_{32}$$

This tells us that in order for a failure to be observed in MISR bit one, the fail location must lie in the union of the critical-path back cone of flops sc_{02}, sc_{20} and sc_{31}. Similarly, the fail locations that can potentially explain the failure in MISR bit two must lie in the union of the

critical-path back cone of flops sc_{01}, sc_{12}, sc_{10}, sc_{21} and sc_{32}. Now based on the single location at a time [16] paradigm, the fail location that explains the observed errors in *both* the MISR bits one and two must be in the intersection of the fail locations found above for the individual failing bits. This is illustrated in Figure 4.

Once the initial candidate location list has been determined as described above, fault simulation can be used to determine the exact locations that describe the observed failure. Again in this case the error function is used to map the fault behaviour captured in the scan flops to the failing MISR bits. This can be done by simply plugging in the values of e_{ij} obtained from fault simulation in the error functions $\varepsilon_k(\mathbf{e})$ to get the corresponding failing MISR bits.

Hence the entire signature based diagnosis process for logic BIST can be summarized in the following steps:

Step 1 Determine the error functions for each MISR bit in the design.

Step 2 Read in the failing test response for a failing chip and record the failing MISR signatures for each failing pattern.

Step 3 For each failing pattern determine the initial candidate fail locations using the error functions of erroneous MISR bits in the failing pattern.

Step 4 Perform fault simulation on this initial list of candidate locations to get the exact locations that explain the erroneous MISR bits in the failing pattern. The error functions are again used in this step. Repeat for all failing patterns to get a list of all fail locations that explain one or more failing patterns.

Step 5 Analyze the list of fail locations determined in Step 4 to identify the locations that are most likely sites of the real defect in the faulty chip, and, further classify the defect as being bridge, open etc.

In the next section results of experiments using the above described techniques are presented.

4. Experimental Results

Experiments were conducted to verify the effectiveness of signature based diagnosis. The basic idea is to inject a small set of stuck-at faults into the logic under test and then perform fault diagnosis. To obtain the fault candidates, first fault simulation is performed with one thousand random patterns generated by PRPG. The detected fault list and the random pattern set are stored. About two thousand faults are randomly sampled from the fault list and these become the fault candidates. Each fault candidate is injected into the circuit and simulated with the random pattern set to determine the faulty signature. The simulation result is compared with the golden signature from good machine simulation and a failure log is generated. Signature based diagnosis process is then performed for each failure log with corresponding pattern set.

We also compare the signature based diagnosis with bypass mode diagnosis. In bypass mode, the scan cell values are directly shifted out from scan chain outputs and full observability for each scan cell is obtained. This is essentially the scan pattern based diagnosis. The same fault list and pattern set are utilized as in the case of signature based diagnosis.

It is worth noting, that a more comprehensive cause-effect analysis is possible [17] such that one can identify stuck, bridge, open, or transition defects. However, the identification of different defect behaviors is mainly based on the good machine values at fault sites regardless using MISR or not. Therefore we only use stuck-at-faults to simplify the comparison. A *diagnostic resolution* is chosen as a figure of merit and is defined here as a reciprocal of the number of suspects. For example, if the call out has only one suspect, the resolution is 100%. If the call out has two suspects, the resolution drops to 50%.

Table 2: General Information of Two Industrial Circuits

circuit	Number of gates	scan cells	Scan chains
CKT1	1.3M	64K	200
CKT2	1.9M	128K	260

Two industrial circuits are used for the experimental purpose. The general design information on these two circuits is described in Table 2. Both designs have over one million gates. The number of scan cells and the number of scan chains present a rough picture of scan chain configuration in logic BIST. It should be noted that in logic BIST mode, all the X sources are bounded to avoid signature corruption. The same X-bounded netlist is used in bypass mode diagnosis.

Table 3 presents the results for signature based diagnosis as well as bypass mode diagnosis. In signature based diagnosis, a 64-bit MISR is used for both circuits. In diagnostic process, the average diagnosis CPU time is recorded as *Ave. runtime*. It should be pointed out that run time is collected on a distributive computing environ-

ment. Multiple diagnosis jobs are executed simultaneously on different processors in a heterogeneous grid network of Linux machines. Due to variance of processor speed, run time collected for diagnosis procedure is not exactly comparable to each other. Nevertheless, the average run time can still give us some indication on how performance is affected with signature based diagnosis. Diagnostic resolution is computed for each case and then average out through all cases in the row *Ave. resolution*. T*otal fail pats*. represent the total number of failing patterns across all sampled faults.

Table 3: Diagnosis Results

circuit	CKT1		CKT2	
mode	signature	bypass	signature	bypass
Faults	2000	2000	2088	2088
Ave. runtime	84.9s	30.4s	71.0s	37.5s
Total Fail pats.	20577	20581	23162	23162
Ave. resolution	73.6%	75.8%	80.7%	83.1%

For the experiments in Table 3, all the faults injected are correctly diagnosed as the top candidate in suspects list. On average, the resolution with signature based diagnosis decreases by about 2-3% compared to that with bypass mode diagnosis. And the average run time per fault diagnosis increases about 2 to 3 times. Considering the simplified flow of signature based diagnosis, the decrease in performance and resolution is very moderate.

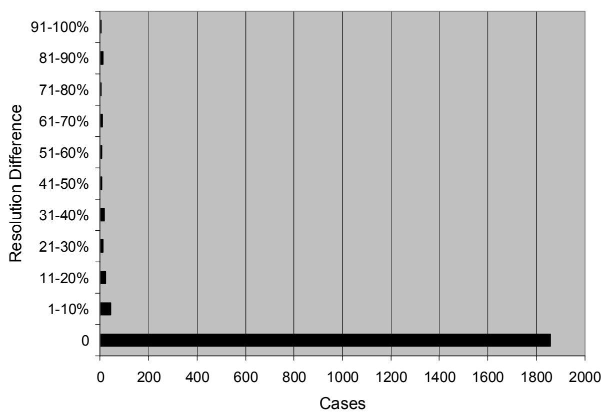


Figure 5: Histogram of Resolution Difference for CKT1

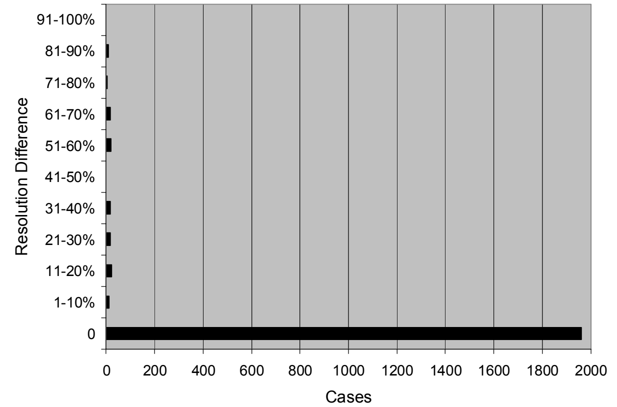


Figure 6: Histogram of Resolution Difference for CKT2

It is interesting to note that for all the sampled faults, the total number of failing patterns is extremely similar for both diagnosis schemes. This reveals the fact that the aliasing probability of a MISR is very low. In Figure 5 and Figure 6, the histograms of resolution difference are plotted for all the diagnosis cases in CKT1 and CKT2. For each diagnosis case, resolution difference is computed as the resolution with bypass mode diagnosis subtracted by the resolution with signature based diagnosis. In the plots, axis Y denotes the range of resolution difference; axis X is the number of cases whose resolution difference falls into corresponding range. For the majority of cases, the diagnosis resolution is the same for both methods. But for a few of them, the resolution of signature based diagnosis may decrease up to 90%. This is understandable due to the fact some information is lost with MISR compaction.

In Table 4, the impact of MISR size is investigated on CKT1. Eight different MISR sizes are experimented. A simple design of space compactor is used. For example, let us consider a logic BIST implementation with 8 scan chains and 2-bit MISR. The space compactor is constructed such that scan chain outputs $\{2n+i\}$ are XORed together and connected to MISR bit i. That is, chains 1, 3, 5, 7 are XORed together and fed into MISR bit 1. Chains 2, 4, 6, and 8 are XORed and connected to MISR bit 2. Whenever a MISR with a different size is implemented, the space compactor is adjusted according to the rule described above. For the impact of MISR size, four numbers are compared. The first one is the total number of failing patterns across all diagnostic cases. The second one is the average resolution. The third one is the average number of terms in error function (Recall from Section 3 that the number of terms in the error function of a MISR bit is equal to the number of scan cells on which the MISR bit depends on). The last one is the average run time per diagnostic case. For comparison, the corresponding results for bypass mode diagnosis are also listed. As

far as resolution is concerned, there is no explicit conclusion that can be drawn from the results. The difference is very marginal. In fact, a 32 or 64-bit MISR serves pretty well for million-gate designs. On the other hand, the average terms of error function tends to decrease with increasing MISR size. This is mainly due to the fact that fewer error bits are mapped to each MISR bit with increasing MISR size. As is shown, a large number of terms in error function do not have much impact on diagnostic performance as is shown by average run time. The variance in the run time is mainly caused by the uncertainty of our distributive computing environment.

Table 4: Impact of MISR Size (on CKT1)

MISR size	Total Failing Patterns	Average Resolution	Average Terms in Error Function	Average Run Time
16	20577	73.3%	33298	81.4s
32	20577	73.5%	25983	66.5s
48	20577	73.5%	26159	86.1s
64	20577	73.6%	19769	84.9s
80	20577	73.7%	15784	107.7s
96	20577	73.7%	15048	124.3s
112	20577	73.6%	8449	67.0s
128	20577	73.6%	6848	67.3s
bypass	20581	75.8%	1	30.4s

5. Conclusion

Experimental results have shown that it is possible to perform diagnostics directly from the failing MISR signatures. This approach requires only two statically generated vector sets (single MISR unload for Phase 1, and MISR unload/reload for each pattern). This greatly simplifies the logic BIST diagnostics flow by eliminating the need to generate and apply DUT specific test vectors. Our results show that this can be accomplished with a minimal impact on diagnosis resolution.

References

[1] C. Eddleman, N. Tamarapalli and W.-T. Cheng, "Advanced Scan Diagnosis Based Fault Isolation and Defect Identification for Yield Learning," *International Symposium for Testing and Failure Analysis* 2005, session 24

[2] C. Stroud, "Automated BIST for Sequential Logic Synthesis," *IEEE Design and Test of Computers* 1988

[3] B. Koenemann, T. Mucha and G. Zwiehoff, "Built in Logic Block Observation Techniques," *Proceedings of the International Test Conference*, 1979

[4] W. H. McAnney and J. Savir, "There is Information in Faulty Signatures," *Proceedings of the International Test Conference*, 1987

[5] P. Wohl, J. Waicukauski, S. Patel and G. Maston, "Effective Diagnostics Through Interval Unloads in a BIST Environment," *Proceedings of the Design Automation Conference* 2002

[6] Y. Wu and S. Adham, "Scan Based BIST Fault Diagnosis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 18, No 2 February 1999

[7] J. Rajski and J. Tyszer, "Diagnosis of Scan Cells in BIST Environment," *IEEE Transactions on Computers* Vol 48, No 7 July 1999

[8] J. Ghosh-Dastidar and N. Touba, "A Rapid and Scalable Diagnosis Scheme for BIST Environments with a Large Number of Scan Chains," *Proceedings of the VLSI Test Symposium* 2000

[9] R. Kapur, T. Williams and M. Mercer, "Directed-Binary Search in Logic BIST Diagnostics," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition* 2002

[10] J. Ghosh-Dastidar, D. Das and N. Touba, "Fault Diagnosis in Scan Based BIST Using Both Time and Space Information," *Proceedings of the International Test Conference* 1999

[11] J. Savir and W. H. McAnney, "Identification of Failing Tests With Cycling Registers," *Proc. Intl. Test Conf.*, 1988, pp. 322-328.

[12] W.-T. Cheng, K.-H. Tsai, Y. Huang, N. Tamarapalli and J. Rajski "Compactor Independent Direct Diagnosis," *Proc. Asian Test Symposium*, pp.204-209, 2004.

[13] A. Leininger, P. Muhmenthaler, W.-T. Cheng, N. Tamarapalli, W. Yang, K.-H. Tsai, "Compression-Mode Diagnosis Enables High-Volume Monitoring Diagnosis Flow", *Proc. International Test Conference*, 2005, paper 7.3.

[14] J. Savir, "Salvaging Test Windows in BIST Diagnostics," *IEEE Transactions on Computers* Volume: 47 Issue: 4 April 1998

[15] P.H. Bardell, W.H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudo Random Techniques*, Wiley Interscience, New York, 1987

[16] T. Bartenstein, D. Heaberlin, L.Huisman and D. Sliwinski, "Diagnosing Combinational Logic Designs using the Single Location At-A-Time(SLAT) Paradigm," in *Proc. Intl. Test Conf.,* 2001, pp. 287-296.

[17] W. Zou, W.-T. Cheng, S. M. Reddy, and H. Tang, "On Methods to Improve Location Based Logic Diagnosis," *Proc. VLSI Design*, 2006, pp. 181-187

[18] S. B. Akers, B. Krishnamurthy, S. Park and A. Swaminathan, "Why is less information from logic simulation more useful in fault simulation" in *Proc. Intl. Test Conf.,* 1990, pp. 786-800.

[19] B. Keller and T. Bartenstein, "Use of MISR Output Streams for Diagnostics" in *Proc. Intl. Test Conf.,* 2005, pp. 735-743.