IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

# STORAGE-BASED BUILT-IN SELF-TEST FOR GATE-EXHAUSTIVE FAULTS

**[1]Ms.Sanju Thasneem, [2]Mr.Gregorious Jose C**

[1]Student, [2]Assistant Professor

[1]Department of Electronics and Communication Engineering

[1]IES College of Engineering

Thrissur-Kerala, India

***Abstract*:** Built-in self-test (BIST ) approaches are suitable for in-field testing since they do not require a tester for storage and application of test data. They also reduce the security vulnerabilities associated with loading and unloading of external test data into scan chains. As technologies evolve, in-field testing needs to address more complex defect and fading mechanisms that require specific deterministic tests. This can be addressed by BIST approaches that store test data on-chip, and use the data for on-chip generation of both random and deterministic tests. In this case, there is a tradeoff between the amount of stored test data and the comprehensiveness of the test set that can be applied. This explores this tradeoff in a specific context that has the following main features. The initial stored test data is based on a stuck-at test set. The target faults are single-cycle gate-exhaustive faults. The stored test data is enhanced gradually by test data based on a gate-exhaustive test set to increase the coverage of gate-exhaustive faults.

***Index Terms*: Built-in-self test, Data on-chip, Single cycle faults, Gate exhaustive test set**

## I. INTRODUCTION

Built-in self-test (BIST) approaches are suitable for in-field testing since they do not require a tester for storage and application of test data. They also reduce the security vulnerabilities associated with loading and unloading of external test data into scan chains. As technologies evolve, in-field testing needs to address more complex defect and fading mechanisms that require specific tests. This can be addressed by BIST approaches that store test data on-chip, and use the data for on-chip generation of both random and deterministic tests. Different from hybrid approaches that combine test data compression with on-chip test generation, a BIST approach stores all the test data on-chip. This is the type of approach considered here. In this approach, there is a tradeoff between the amount of stored test data and the comprehensiveness of the test set that can be applied. Here explores this tradeoff in the following context. The circuit under consideration has n scan chains of length l. One component of the stored test data is a set S(i) of scan vectors. Using only scan vectors from S(i), an on-chip test generation logic applies a fixed number of tests referred to as random, where scan vectors are selected randomly from S(i), and a small number of tests referred to as deterministic that consist of specific scan vectors from Si. The set of deterministic tests is denoted by T (i, deterministic), and it requires additional storage of test data (indices of scan vectors from S(i) that form the tests in T (i, deterministic). The test set produced on-chip is denoted by Ti, and it changes with S(i) and T (i, deterministic). The on-chip test generation logic is the same for every S(i) and T (i, deterministic), and every circuit. Only its parameters (e.g., memory and multiplexer sizes) are circuit-dependent. A deterministic test set for single stuck-at faults is used as an initial source for test data. The test data is a set S0 of scan vectors based on test set, and a set T (0, deterministic) of deterministic tests to complement the random tests based on S0. With S0 and T (0, deterministic), and the on- chip test generation logic, T0 detects all the detectable single stuck- at faults. The set of single stuck-at faults. The effectiveness of the random tests is enhanced by using stored scan vectors from test set. In addition to stuck-at faults, the set of target faults includes single-cycle (static) gate-exhaustive faults. These are used for representing the need to cover a large number of defects with specific actions that are more difficult to detect than stuck-at faults. Gate- exhaustive faults are suitable for this purpose since their number is large, and they require more values to be assigned by a test. The set of detectable gate-exhaustive faults is denoted by F (gate exhaustive), and a deterministic test set for gate-exhaustive faults is denoted by T gate exhaustive. It is also possible to consider a test set for transition faults as an initial source for test data, and two-cycle (dynamic) gate-exhaustive faults as additional target faults. When the pair (S0, T0, deterministic) is used for on-chip test generation, the coverage of gate-exhaustive faults may not be complete. An iterative software procedure produces a series of pairs (S0, T0, deterministic), (S1, T1, deterministic), ..., (Sm−1, Tm−1, deterministic). For $i > 0$, the procedure obtains S(i) from S (i−1) by adding at least one new scan vector based on T gate exhaustive. The scan vectors added to S(i) allow the coverage of gate- exhaustive faults to be increased when the on-chip test generation logic is used for S(i), Ti, deterministic). With (Sm−1, Tm−1, deterministic), complete coverage of gate-exhaustive faults is achieved. In many cases, the increased fault coverage is achieved by random tests that use scan vectors from S(i), and T (i, deterministic = ∅).

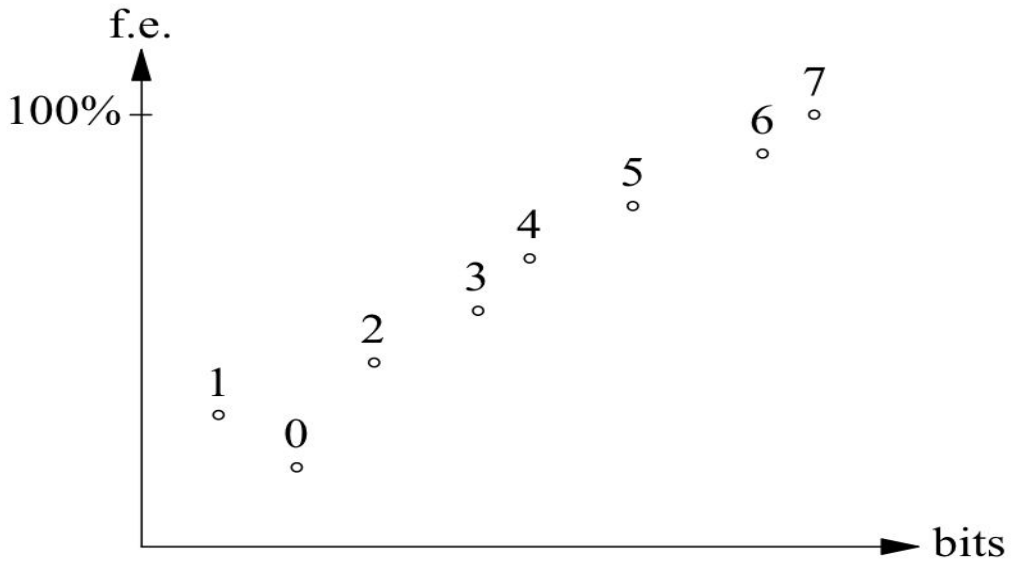



Fig. 1. Tradeoff between fault efficiency and stored test data.

Figure 1 illustrates the tradeoff explored by the software procedure that produces the pairs (S(i), Ti, deterministic) for i ≥ 0. Figure 1 is based on the results obtained later for benchmark circuit s1423. A circle with i above it corresponds to a test set Ti. The horizontal axis in Figure 1.1 shows the number of storage bits for Ti. The vertical axis shows the fault efficiency, which is the percentage of detected faults out of the detectable faults, achieved for gate-exhaustive faults by Ti. The fault efficiency reaches 100% in iteration i = 7. In Figure 1.1, when the pair (S1,T1, deterministic) is computed in iteration 1, it has a lower number of storage bits than (S0,T0, deterministic), and a higher fault efficiency. Therefore, the solution obtained in iteration 1 is preferred over the solution obtained in iteration 0. Overall, there are seven viable solutions in Figure 1, corresponding to (S(i), Ti, deterministic) for i = 1, 2, 3, 4, 5, 6 and 7. The approach described is developed for the case where a circuit has a large number of scan chains, making the scan chain length small enough for storage of scan vectors to be feasible. A large number of scan chains is also used by test data compression approaches to control the test application time. The is arranged as follows. The on-chip test generation logic is described in Section II. A software procedure for computing the sets S(i) and T(i),deterministic is described . Experimental results for benchmark circuits are presented in next chapters.

## II. RELATED WORK

This paper [1] described a built-in test pattern generation method for scan circuits. Under this method, a precomputed test set is partitioned into several sets containing values of primary inputs or state variables. The sets are stored on-chip and the on-chip test set is obtained by implementing the Cartesian product of the various sets. The sets are reduced as much as possible before they are stored on-chip in order to reduce the storage requirements and the test application time. Also describe two schemes for reducing the set sizes, one where each set stores the values of one subset of primary inputs or state variables and one where a single set is used to store values of different subsets of state variables. Demonstrate the effectiveness of the proposed method as a stand-alone procedure and as part of a scheme where random patterns are first applied to detect easy-to-detect faults. In the latter case, the proposed method is applied to detect the hard-to-detect faults that remain undetected.

The conventional test-per-scan built-in self-test (BIST) scheme needs a number of shift cycles followed by one capture cycle. Fault effects received by the scan flip-flops are shifted out while shifting in the next test vector, like scan testing. Unlike deterministic testing, it is unnecessary to apply a complete test vector to the scan chains. A new scan-based BIST scheme is proposed by properly controlling the scan enable signals of the scan chains. Different weighted values are assigned to the scan enable signals of scan flip-flops in separate scan chains. Capture cycles can be inserted at any clock cycle if necessary. A new testability estimation procedure [2] according to the proposed testing scheme is presented. A greedy procedure is proposed to select a weight for each scan chain. Experimental results show that the proposed method can improve test effectiveness of scan-based BIST greatly and most circuits can obtain complete fault coverage or very close to complete fault coverage.

There is a growing acceptance in the industry that with 0 1 ~ 3 processes and beyond, at-speed structural testing is the only viable methodology for achieving acceptable quality levels with acceptable yields. Fully embedded at-speed structural test approaches have been growing in use over the past years. For embedded memories, fully embedded test approaches are now widespread in use. For random logic testing, three approaches are currently in use. The full scan ATPG methodology provides a fully outward test approach. The more recent ATPG based procedures represent a hybrid approach consisting of some inward IP as well as reduced outward scan test data. Finally, the logic BIST methodology represents a fully embedded test approach requiring no outward test data. There is growing evidence that fully embedded test approaches for logic can provide increased product quality (lower DPM levels) while maintaining or lowering manufacturing test costs over existing external or even hybrid approaches. In addition to these benefits, embedded test has the unique advantage of being fully portable and reusable. Most designs today are designed hierarchically, with cores often being designed by different teams or even acquired from 31d party providers. With an embedded test approach, a block or core can be made fully self-testable, with a standard interface ( e.g. IEEE P1500) for accessing the embedded test capabilities. This not only provides test cost savings when the core is reused either within the design or across

different designs, but also provides two distinct security benefits. The first of these benefits is IP protection. Because the core does not need to be processed for test insertion, it is possible to deliver the core as a black box. In additional since all test pattern data and results are generated and examine on chip, there is no need to provide any stimulus or expect test data along with the core. This is tum removes any possibility of using this data to reverse engineer the core design. The other security issue is access to sensitive data stored inside the device once it is active in the field. Any external access to embedded memories or inward functional registers represents such a security risk. With embedded testing approaches, external access to either memories or inward functional registers is not typically required for production testing. External access is only provided to the embedded test IP for run time and failure result extraction. Although the failure result extraction process may result in the need to provide some access to inward functional registers, this access can be made dependent on embedded test IP access instructions. These instructions are typically (or can be made to be) 32 to 64 bits long and therefore provide a secure entry point. In summary, security is simply another factor which can best be addressed by fully embedding test and diagnostic abilities within the silicon.

Electronic design of high-performance digital systems in nano-scale CMOS technologies under Process, power supply Voltage, Temperature and fading (PVTA) variations is a challenging process. Such variations induce abnormal timing delays leading to systems errors, harmful in safety-critical applications. Performance Failure Prediction (PFP), instead of error detection, becomes necessary, particularly in the presence of fading effects. In this paper (4), an on-line BIST methodology for PFP in a standard cell design flow is proposed. The methodology is based on abnormal delay detection associated with critical signal paths. PVTA-aware fading sensors are designed. Multi- level simulation is used. Functional and structural test pattern generation is performed, targeting the detection of critical path delay faults. A sensor insertion technique is proposed, together with an up-graded version of a proprietary software tool, DyDA. Finally, a novel strategy for gate-level fading fault injection is proposed, using the concept of fading de-rating factor. Results are presented for a Serial Parallel Interface (SPI) controller, designed with commercial UMC 130nm CMOS technology and Faraday TM cell library. Only seven sensors are required to monitor unsafe performance operation, due to Negative Bias Thermal Instability (NBTI)-induced fading [3].

Traditionally, analog devices are tested based on parametric measurements wherein the measured parameters are compared with designer-defined specifications and a pass/fail decision is made. While this has been effective, there are several issues with the traditional approach that require a new way of thinking. First, this approach [4] results in high engineering and test application costs, requiring long test development cycles, long test times, and expensive test equipment. Second, some analog Devices Under Tests (DUTs), such as DC/DC converters, data converters, and sensor front-ends, are not easily accessible from the pinouts. While these devices are generally tested as parts of an entire system, relying on system-level tests increases the likelihood of missing small or latent defects and stability-related problems that arise later in the field or under different input/environment conditions. Here proposed a fully differential, correlated zoom-ADC architecture with a passive loop filter for low- frequency Built-In Self-Test (BIST) applications, along with a synthesis tool that can target various design specifications. Here present the detailed ADC architecture and a step-by-step process for designing the zoom-ADC. The design flow does not rely on the extensive knowledge of an experienced ADC designer. Two ADCs have been correlated with different performance requirements in the 65nm CMOS process. The first ADC 3 achieves a 90.4dB Signal-to-Noise Ratio (SNR) in 512µs measurement time and consumes 17µW power. The second design achieves a 78.2dB SNR in 31.25µs measurement time and consumes 63µW power.

## III. ON-CHIP TEST GENERATION

The scan configuration assumed in this paper has n scan chains. For simplicity it is assumed that all the scan chains have the same length, l. This can be achieved by padding each scan chain until its length reaches l. Padding is not needed for the physical circuit, only for the model used by the software procedure to compute the test data for the on-chip test generation logic.

One component of the test data stored on-chip is a set of scan vectors, S(i) = {s0, s1, ..., s v−1}. For illustration, Table I(a) shows the set S0 obtained for a circuit with n = 3 scan chains of length l = 3.

A test t(j) is formed by selecting n scan vectors from S(i), one for every scan chain. With scan vectors s j0 , s j1 , ..., s j(n−1) , we have that t(j) = ⟨s (j0),s (j1),...,s (j (n−1))⟩. For $0 \leq k < n$, the scan vector for scan chain k is denoted by t (j, k) = s (jk).

Table I(b) shows a test set T0 obtained for the circuit from Table I(a). For every test t(j) , Table I(b) shows the indices j0 , j1 , j2 , and the subtests t (j,0), t (j,1) and t (j,2). The variable πj is explained next.

The on-chip test generation logic applies two types of tests. A random test t (j)= ⟨s (j0),s (j1), ..., s (j (n−1)) ⟩ is designated by πj = 0. In this case, for $0 \leq k < n$, the scan vector s (jk) for the subtest t (j, k) is selected randomly.

A deterministic test t (j) = ⟨s (j0),s (j1), ..., s (j(n−1))⟩ is designated by πj = 1. In this case, the set of indices j0, j1, ..., j (n−1) is stored in an on-chip memory.

TABLE I

Example Test Set

(a) Scan Vectors

| $p$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $s_p$ | 011 | 110 | 000 | 101 | 010 | 001 |

(b) Test Set

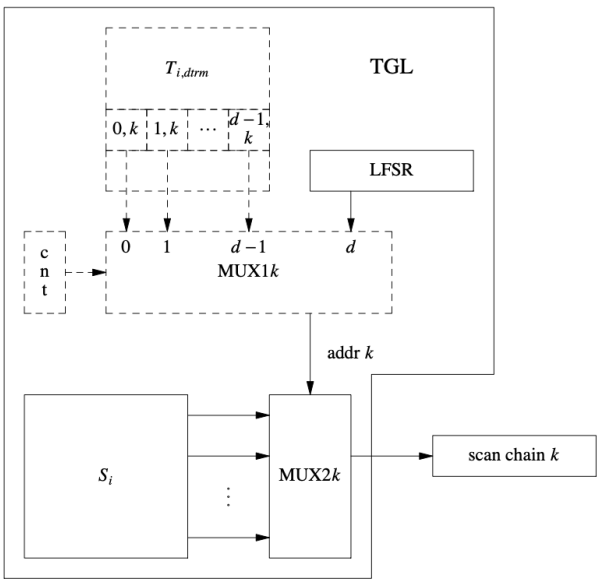
| $j$ | $\pi_j$ | $j_0$ | $j_1$ | $j_2$ | $t_{j,0}$ | $t_{j,1}$ | $t_{j,2}$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 4 | 1 | 5 | 010 | 110 | 001 |
| 1 | 0 | 1 | 1 | 3 | 110 | 110 | 101 |
| 2 | 0 | 2 | 1 | 5 | 000 | 110 | 001 |
| 3 | 0 | 2 | 2 | 0 | 000 | 000 | 011 |
| 4 | 1 | 5 | 4 | 5 | 001 | 010 | 001 |
| 5 | 1 | 5 | 5 | 5 | 001 | 001 | 001 |

The on-chip test generation logic is illustrated by Figure 2. The lower part of Figure 2 shows the memory storing the set of scan vectors Si. The size of the memory is v×l. It also shows scan chain k of length l. A multiplexer called MUX2k selects one of the scan vectors from S (i) depending on the variable called adder k. The number of bits in adder k is log2(v). The selected scan vector is scanned into scan chain k. For a test t(j) it defines the subtest t (j, k).

The value of adder k is computed by the logic in the upper part of Figure 2. If only random tests are applied, the dashed part of Figure 2 is not needed, and an LFSR produces the values for adder k. In general, the deterministic part of the test set Ti, T (i, deterministic), is stored in a memory of size d × n × log2(v), where d is the number of deterministic tests. The entries inside T (i, deterministic) in Figure 2 correspond to scan chain k. These are indices of scan vectors that need to be loaded into scan chain k under the deterministic tests t0 , t1 , ..., td−1 .

A counter denoted by cnt determines which test is applied through a multiplexer denoted by MUX1k. A count value between 0 and d−1 corresponds to a deterministic test from T (i, deterministic). A count value of d corresponds to a random test. In this case, the LFSR provides log2(v) bits selecting a scan vector randomly from Si. The counter stays at d to apply R random tests, for a parameter R.

The memories S(i) and T (i, deterministic), as well as the counter and LFSR, are common to all the scan chains. In the case of the LFSR, each scan chain uses a distinct subset of bits to obtain a different random number. In addition, each scan chain requires two multiplexersThe overall storage requirements for the two memories are v · l + d · n · log2(v) bits. The memories dominate the size of the on-chip test generation logic




.

Fig. 2. On-chip test generation logic.

The entire test generation logic, including both multiplexers for every scan chain, resides close to the memories, within the outline marked TGL in Figure 2. Each scan chain is driven by a single line, represented by the output of MUX 2k in Figure 2. The routing overhead is similar to that of test data decompression logic that drives all the scan chains.

For the output response it is assumed that sequential output compaction will be performed by output compaction logic such as a multiple-input shift-register (MISR).

## IV. PROCEDURE FOR COMPUTING STORED TEST DATA

This section describes an iterative software procedure for computing sets of scan vectors S(i) and deterministic tests T(i, deterministic) for on-chip test generation. For simplicity of discussion, S(i) is associated with a test set Ti that consists of both random and deterministic tests. The non-random tests in Ti define T(i, deterministic).

### 4.1. Overview

The procedure accepts a test set for single stuck-at faults, and a test set T(gate exhaust) for gate-exhaustive faults. It produces scan sets vectors S0, S1, ..., Sm−1, with test sets T0, T1, ..., Tm−1.

At the beginning of iteration i = 0, the procedure initialises S0 based on test set, as follows. All the distinct scan vectors of test set are included in S0, and T0 = test set initially. With this initialisation, every test in T0 can be expressed in terms of scan vectors from S0.

At the beginning of iteration i > 0, S(i) = S(i−1) and Ti = Ti−1. At the end of iteration $0 \leq i \leq m-2$, the procedure adds at least one scan vector to Si. It adds tests that use the new scan vectors to Ti. The procedure terminates when, at the end of iteration $m-1$, it does not add any scan vectors to Sm−1An arbitrary iteration i proceeds as described next. The procedure referred to as Procedure 0 is applied first to remove unnecessary scan vectors from Si. The test set Ti is modified to ensure that it uses only scan vectors from S(i) .

The procedure referred to as Procedure 1 first stores the current test set T(i) in a test set denoted by T (can d), and initialises Ti to be empty. All the target single stuck-at faults are included, and all the target gate-exhaustive faults are included in F(gate exhaustive).

For a parameter R, Procedure 1 includes R random tests in Ti. For every test it performs fault simulation with fault dropping of fault set and F(gate exhaustive). It then uses deterministic tests from T (can d) to detect additional faults from fault set and F(gate exhaustive) using only scan vectors from Si.

All the detectable stuck-at faults from fault set are guaranteed to be detected by Ti after Procedure 1 is applied. The procedure terminates if all the detectable gate-exhaustive faults from F(gate exhaustive) are detected by Ti as well. Otherwise, Procedure 2 adds to Ti a limited number of additional deterministic tests based on T(gate exhaustive) to detect additional gate-exhaustive faults. As much as possible, Procedure 2 uses only scan vectors that are already included in Si. When this is not possible, Procedure 2 adds to Ti tests that require new scan vectors, which are added to Si. The procedure prefers tests that require the smallest possible numbers of new scan vectors. As Ti is modified, the procedure maintains fault detection in- formation for Ti. Fault detection information is obtained by fault simulation with fault dropping fault set ∪ F(gate exhaustive) under Ti. For a fault f ∈ fault set ∪ F(gate exhaustive), the first test in Ti that detects it is denoted by t deterministic (f) . For an undetected fault, deterministic (f) = −1. Fault simulation with fault dropping is also applied at the end of every procedure.

### 4.2. Procedure 0

For Procedure 0 all the tests in Ti are considered as deterministic. Procedure 0 associates with every scan vector s(p) ∈ S(i) the number of times it is used by a test in Ti. This number is denoted by a(s(p)). The procedure considers the scan vectors by increasing order of a (s(p)). This is based on the expectation that it will be easier to remove from S(i) a scan vector that is used fewer times by tests from Ti. The procedure maintains the variables a(s(p)) up-to-date as it modifies the test set. A scan vector s(p) with a(s(p)) = 0 is removed from Si.

When the procedure considers s(p) for removal, let the subset of tests where s(p) appears be T(s(p)). Let the subset of faults that the tests in T(s(p)) detect be F(s(p)). To facilitate the modification of the tests in T (s(p)), the procedure simulates F (s(p)) under Ti \ T (s(p)). If a fault f ∈ F (s(p)) is detected by a test t(j) ∈ Ti \ T (s(p)), the procedure assigns deterministic (f) = j, and removes the fault from F(s(p)).

The procedure considers every subtest t(j, k) such that t(j, k) = s(p) separately. For t(j, k), it considers as an alternative every scan vector s(q) ∈S(i) such

that a(s(q)) = 0 and q≠p . To check whether s(q) is an acceptable alternative, the procedure assigns t (j, k) = s(q). It then simulates every fault f ∈ fault set ∪ F)(gate exhaustive such that deterministic(f) = j under the modified t(j). The modification of t(j) is accepted if all the faults with deterministic(f) = j are detected. Otherwise, the procedure reassigns t(j, k) = sp. If none of the alternatives of s(p) is acceptable, the procedure concludes that s(p) cannot be removed, and it does not consider other subtests where s(p) appears.

The procedure considers the tests in T(s(p)) by decreasing order of the number of faults they detect. This is based on the expectation that tests with larger numbers of detected faults are more difficult to modify. If such a test cannot be modified, the procedure will not consider other tests in T(s(p)). The procedure considers the replacement scan vectors s(q) by increasing order of the number of appearances in Ti , a(s(q)). This is based on the expectation that a more uniform use of scan vectors allows faults to be detected more uniformly, and makes it easier to modify the test set.

After considering all the scan vectors in S(i) for removal, if any scan vector was removed from S(i), the procedure performs another pass over the scan vectors in S(i), in case additional scan vectors can be removed after modifying the test set. Procedure 0 terminates after a pass that does not reduce the number of scan vectors in Si.

### 4.3. Procedure 1

Procedure 1 initially assigns T(can d) = Ti, Ti = ∅, and includes all the target faults in fault set and F(gate exhaustive).

Procedure 1 includes R random tests in Ti, for a parameter R. A random test t(j) is constructed by selecting t(j, k) randomly from S(i) (or using the LFSR in Figure 2), for $0 \leq k < n$. The procedure performs fault simulation with fault dropping t(j) under fault set ∪ F(gate exhaustive), for every random test t(j).

Next, the procedure uses tests from T(can d) as deterministic tests to detect additional faults from fault set ∪ F(gate exhaustive). The goal is to ensure that both the stuck-at and gate-exhaustive fault coverages of Ti are not lower than those of T(can d). This ensures that the stuck-at fault coverage is maintained, and the gate-exhaustive fault coverage increases monotonically with every iteration until all the detectable gate-exhaustive faults are detected in the last iteration. For every test t(j) ∈ T(can d), the procedure applies the following steps.

The procedure simulates fault set ∪ F(gate exhaustive) under t(j). If no faults are detected, the procedure does not consider t(j) further. Otherwise, it attempts to modify t(j) to increase the number of faults it detects out of fault set ∪ F(gate exhaustive). For this purpose, the procedure includes in F(t(j)) all the faults that t(j) detects. It then considers every scan chain $0 \leq k < n$, and every scan vector s(p) ∈ S(i). If t(j, k) = s(p), the procedure defines a test t mod that is equal to t(j), except that t mod = sp. It simulates the faults in F(t(j)) under t mod. If all the faults are detected, (j, k) it also simulates the faults in fault set ∪ F(gate exhaustive) under t(j). Let the subset of detected faults be F(t mod). The procedure accepts the modification of t(j) if F(t(j)) ⊆ F(t mod). In this case, it assigns t(j) = t mod and F(t(j)) = F(t mod). Otherwise, it discards t mod.

If the number of faults detected by t(j) was increased, the procedure performs another pass over all the scan chains and all the scan vectors in Si. The final test tj is added to Ti, and the faults it detects are removed from fault set and F(gate exhaustive).

After considering every test t(j) ∈ T(can d), Ti detects all the faults from fault set ∪ F(gate exhaustive) that T(can d) detects, and possibly additional faults from F(gate exhaustive).

### 4.4. Procedure 2

Procedure 2 adds to Ti a limited number of tests based on T(gate exhaustive). Its goal in selecting which tests will be added is to detect as many additional gate-exhaustive faults as possible using only scan vectors that are already included in S(i), or require the addition of as few new scan vectors to Si as possible. It stops after a limited increase in the gate-exhaustive fault efficiency is achieved to avoid a large increase in the storage requirements in a single iteration. The number of tests from T(gate exhaustive) that the procedure uses depends on the circuit and the iteration. The tests are modified as described below to ensure that as few new scan vectors as possible are added to Si.

Procedure 2 is applied iteratively until it achieves its goal. In each pass Procedure 2, it considers every test t(j) ∈ T(gate exhaustive). When it considers t(j), it first performs fault simulation of F(gate exhaustive) under t(j). If any faults are detected, the procedure continues with t(j) as follows.

It assigns t new = t(j), and includes the faults detected by t(j) in a j set F(t new). It also assigns j new = −1 for $0 \leq k < n$ to indicate jk that the scan vectors of t new may not be included in S . It then ji considers every scan chain $0 \leq k < n$, and every scan vector s(p) ∈ Si. The procedure defines a test t mod that is equal to t new, except that t mod = s(p) and j(k) mod = p. It simulates the faults in F(t ) under t(j mod). If all the faults are detected, it also simulates the faults in F(gate exhaustive under t(j)mod. Let the subset of detected faults be F(t mod). The procedure accepts the modification of t new if F(t new) ⊆ F(t mod). In this case, it assigns t(j)new = t(j)mod and F(t j mod) = F(t j mod). Otherwise, it discard t(j)mod.

If the procedure modified j new = −1 into j new ≥ 0 for at least one scan chain k, it performs another pass over all the scan chains with j new = −1. When modification of t new ends, let n(t new) be the number of detected gate-exhaustive faults, and m(t new) the number of scan chains k for which j new = −1.

If m(t new) = 0, t new uses only scan vectors from S(i), and detects n(t new) gate-exhaustive faults. In this case, the procedure adds t new to Ti, and removes the faults it detects from F(gate exhaustive). Otherwise, the procedure defers the decision on t new to the end of the pass. At the end of the pass, if no test with m(t new) = 0 was added to Ti, the procedure selects the test t new with the smallest value of m(t new), and the largest value of n(t new). It adds to S(i) all the new scan vectors used by t new, i.e., S(i) = S(i) ∪ {t new:0≤k<n, jk = −1}. It also adds t new to Ti.

After adding at least one new scan vector to S(i), Procedure 2 terminates if the gate-exhaustive fault efficiency is increased by a parameter denoted by Δgate exhaustive. This parameter is needed to ensure that the gate-exhaustive fault efficiency increases substantially with every iteration. Procedure 2 also terminates when all the faults in F(gate exhaustive) are detected. Before using Ti in iteration i + 1, the procedure performs forward-looking reverse order fault simulation to remove unnecessary tests from Ti.

## V. EXPERIMENTAL RESULTS

The software procedure for computing the sets S(i) and Ti was applied to benchmark circuits.

The following parameter values were used. For a circuit with K flip-flops, the flip-flops were partitioned into n scan chains such that n2 ≥ K. If necessary, n2 − K flip-flops were added for padding. The length of a scan chain was l = n. This yields a large number of short scan chains. Other configurations with the same property can be used instead.

The required increase in the gate-exhaustive fault efficiency for every iteration, Δgate exhaustive, was set as follows. When the gate-exhaustive fault efficiency after the first application of Procedure 1 is at least 95%, Δgate exhaustive = 0.2%. Otherwise, Δgate

exhaustive = 1%. The circuit name is followed by ".1" in this case. These values prevent the procedure from performing an excessive number of iterations.

When the fault efficiency after the first application of Procedure 1 is lower than 95%, Procedure 2 selects a deterministic test after considering ten tests from Tgexh that detect new faults. This is important for limiting the computational effort of the procedure.

To define gate-exhaustive faults, the circuit was partitioned into two-level sub circuits with at most 10 inputs. Each sub circuit was used as a gate, and all its gate-exhaustive faults were added to F(gate exhaustive). Test generation was carried out to produce the test set T(gate exhaustive). Undetectable faults were eliminated from F(gate exhaustive).

The number of random tests was R = 4000 mod test set. This number was selected based on experimental results indicating that a small number of deterministic tests is typically needed for complementing this number of random tests.

The results for iteration i of the procedure are reported before Procedure 2 adds new scan vectors to Si. Up to this point, the procedure utilises the scan vectors in S(i) for both random and deterministic tests (new scan vectors that are added to S(i) are utilised for random tests only in iteration i + 1).

Referring to Figure 1, not every iteration yields improved results compared with later iterations. The results are reported only for iterations with improved results.

The results are reported in Table II as follows. For most of the circuits in Table II, there is a row for every iteration that yields an improved solution until 100% fault efficiency is reached for gate- exhaustive faults. These circuits are arranged by increasing number of iterations that produce improved solutions. Additional circuits are considered with a single iteration to demonstrate the results achievable for them.

After the circuit name, column K shows the number of flip-flops. Column n shows the number of scan chains. This is also the number of flip-flops in a scan chain.

Column i shows the iteration. Column deterministic shows the number of deterministic tests in T(i, deterministic) Column vector shows the number of scan vectors in Si. Column bits shows the number of storage bits required for S(i) and T(i, deterministic) (sub column tot), and the number of storage bits divided by the number of bits required for storing T(gate exhaustive) (sub column tot/gate exhaustive).

The stuck-at fault coverage of Ti is always equal to the fault coverage of test set, which detects all the detectable stuck-at faults. Column gate exhaustive shows the gate-exhaustive fault efficiency achieved by Ti. Column n time shows the normalised runtime of the software procedure, which is the runtime divided by the runtime for fault simulation with fault dropping of fault set under test set, and F(gate exhaustive) under T(gate exhaustive). Since the software procedure is based on repeated fault simulation, normalising its runtime to fault simulation time provides an indication of its computational effort.

The following points can be seen from Table II. The on-chip test generation logic is able to achieve 100% gate-exhaustive fault efficiency. This is a benefit of storing test data and allowing deterministic tests to be applied.

The number of on-chip stored bits is a small fraction of the number of bits in a deterministic gate-exhaustive test set. This is made possible by the on-chip generation of random tests.

Figure 1 is based on s1423 and demonstrates the tradeoff between the number of storage bits and the fault efficiency obtained for gate- exhaustive faults. A similar tradeoff exists for other circuits in Table II. There is only a small number of circuits for which the best solution is obtained in iteration 0, and additional iterations are not required.

The normalised runtime is that of the software procedure. The procedure is based on fault simulation of large numbers of tests to select the best stored data for the on-chip test generation logic. The normalised runtime is similar for circuits of different sizes, indicating that the procedure scales similar to a fault simulation procedure.

The number of deterministic tests in Table II is typically small, implying that the number of storage bits for deterministic tests is kept low. When it is zero, the logic in the upper part of Figure 2 consists only of an LFSR. The number of deterministic tests can be reduced further by increasing the number of random tests applied based on Si.

The random tests produced by the on-chip test generation logic are different from the conventional random tests in that they are formed by random combinations of stored scan vectors, which are derived from a deterministic test set. The more effective the random tests are for a circuit, the lower the sizes of S(i) and T(i, deterministic). This is independent of other parameters such as the size of the circuit.

TABLE II

EXPERIMENTAL RESULTS

| | | | | | | bits | | gexh | |
|---|---|---|---|---|---|---|---|---|---|
| circuit | K | n | i | dtrm | vect | tot | tot/gexh | f.e. | ntime |
| s35932 | 1764 | 42 | 0 | 0 | 444 | 18648 | 0.392 | 100.000 | 9505.74 |
| b04 | 81 | 9 | 1 | 1 | 30 | 315 | 0.017 | 100.000 | 1394.00 |
| des_area | 400 | 20 | 0 | 0 | 431 | 8620 | 0.057 | 100.000 | 1659.84 |
| sasc | 144 | 12 | 0 | 0 | 112 | 1344 | 0.089 | 100.000 | 1023.12 |
| simple_spi | 169 | 13 | 1 | 0 | 47 | 611 | 0.015 | 100.000 | 8612.06 |
| spi | 289 | 17 | 4 | 1 | 241 | 4233 | 0.008 | 100.000 | 63387.09 |
| systemcdes | 324 | 18 | 0 | 0 | 224 | 4032 | 0.125 | 100.000 | 1753.96 |
| usb_phy | 121 | 11 | 0 | 0 | 43 | 473 | 0.049 | 100.000 | 484.20 |
| i2c | 169 | 13 | 1 | 0 | 60 | 780 | 0.022 | 99.339 | 3217.24 |
| i2c | 169 | 13 | 3 | 2 | 57 | 897 | 0.025 | 99.807 | 7068.20 |
| i2c | 169 | 13 | 4 | 5 | 60 | 1170 | 0.032 | 100.000 | 8998.16 |
| tv80 | 400 | 20 | 1 | 128 | 260 | 28240 | 0.049 | 99.574 | 12073.79 |
| tv80 | 400 | 20 | 2 | 129 | 262 | 28460 | 0.050 | 99.778 | 24388.65 |
| tv80 | 400 | 20 | 4 | 134 | 287 | 29860 | 0.052 | 100.000 | 33120.68 |
| b15 | 484 | 22 | 1 | 1503 | 221 | 269390 | 0.261 | 99.539 | 52786.41 |
| b15 | 484 | 22 | 2 | 1626 | 214 | 290884 | 0.282 | 99.705 | 76645.83 |
| b15 | 484 | 22 | 3 | 1768 | 212 | 315832 | 0.306 | 99.932 | 115207.67 |
| b15 | 484 | 22 | 4 | 1802 | 206 | 321684 | 0.312 | 100.000 | 146831.08 |
| wb_dma | 784 | 28 | 1 | 14 | 209 | 8988 | 0.039 | 99.455 | 11906.35 |
| wb_dma | 784 | 28 | 2 | 17 | 229 | 10220 | 0.044 | 99.660 | 19847.86 |
| wb_dma | 784 | 28 | 3 | 17 | 257 | 11480 | 0.049 | 99.865 | 25788.47 |
| wb_dma | 784 | 28 | 4 | 22 | 266 | 12992 | 0.056 | 100.000 | 30586.44 |
| s5378 | 225 | 15 | 1 | 0 | 159 | 2385 | 0.028 | 98.833 | 3729.10 |
| s5378 | 225 | 15 | 3 | 0 | 171 | 2565 | 0.030 | 99.238 | 9174.20 |
| s5378 | 225 | 15 | 4 | 0 | 179 | 2685 | 0.031 | 99.440 | 10663.98 |
| s5378 | 225 | 15 | 5 | 0 | 183 | 2745 | 0.032 | 99.642 | 13204.55 |
| s5378 | 225 | 15 | 7 | 0 | 193 | 2895 | 0.034 | 100.000 | 16628.05 |
| s9234 | 256 | 16 | 0 | 145 | 303 | 25728 | 0.117 | 98.493 | 2576.53 |
| s9234 | 256 | 16 | 1 | 213 | 232 | 30976 | 0.141 | 99.181 | 8841.67 |
| s9234 | 256 | 16 | 2 | 234 | 222 | 33504 | 0.152 | 99.632 | 14736.02 |
| s9234 | 256 | 16 | 3 | 262 | 224 | 37120 | 0.169 | 99.941 | 20309.77 |
| s9234 | 256 | 16 | 4 | 264 | 221 | 37328 | 0.170 | 100.000 | 25101.02 |
| s1423 | 100 | 10 | 1 | 0 | 53 | 530 | 0.067 | 98.609 | 1241.20 |
| s1423 | 100 | 10 | 2 | 1 | 53 | 590 | 0.075 | 98.851 | 1936.00 |
| s1423 | 100 | 10 | 3 | 1 | 57 | 630 | 0.080 | 99.093 | 2473.00 |
| s1423 | 100 | 10 | 4 | 1 | 59 | 650 | 0.082 | 99.335 | 3153.60 |
| s1423 | 100 | 10 | 5 | 1 | 63 | 690 | 0.087 | 99.577 | 3681.20 |
| s1423 | 100 | 10 | 6 | 1 | 67 | 740 | 0.093 | 99.819 | 4205.00 |
| s1423 | 100 | 10 | 7 | 1 | 69 | 760 | 0.096 | 100.000 | 4861.20 |
| s13207.1 | 729 | 27 | 1 | 0 | 484 | 13068 | 0.026 | 90.974 | 26106.04 |
| s13207.1 | 729 | 27 | 2 | 0 | 498 | 13446 | 0.027 | 91.992 | 40675.93 |
| s13207.1 | 729 | 27 | 3 | 0 | 522 | 14094 | 0.028 | 93.148 | 49713.95 |
| s13207.1 | 729 | 27 | 4 | 0 | 548 | 14796 | 0.030 | 94.176 | 62855.36 |
| s13207.1 | 729 | 27 | 5 | 0 | 586 | 15822 | 0.032 | 95.232 | 73768.11 |
| s13207.1 | 729 | 27 | 6 | 0 | 611 | 16497 | 0.033 | 96.345 | 87076.30 |
| s13207.1 | 729 | 27 | 7 | 0 | 633 | 17091 | 0.034 | 97.420 | 96723.46 |
| s13207.1 | 729 | 27 | 8 | 0 | 648 | 17496 | 0.035 | 98.444 | 103565.09 |
| s13207.1 | 729 | 27 | 9 | 0 | 714 | 19278 | 0.039 | 99.472 | 112481.73 |
| s13207.1 | 729 | 27 | 10 | 0 | 737 | 19899 | 0.040 | 100.000 | 122845.27 |
| s15850.1 | 625 | 25 | 0 | 3 | 288 | 7875 | 0.013 | 86.801 | 1509.86 |
| s15850.1 | 625 | 25 | 1 | 32 | 349 | 15925 | 0.025 | 88.121 | 5625.45 |
| s15850.1 | 625 | 25 | 2 | 60 | 433 | 24325 | 0.039 | 89.297 | 10849.58 |
| s15850.1 | 625 | 25 | 3 | 46 | 521 | 24525 | 0.039 | 90.465 | 19150.70 |
| s15850.1 | 625 | 25 | 4 | 59 | 607 | 29925 | 0.048 | 91.554 | 25021.39 |
| s15850.1 | 625 | 25 | 5 | 75 | 699 | 36225 | 0.058 | 92.622 | 31400.64 |
| s15850.1 | 625 | 25 | 7 | 72 | 908 | 40700 | 0.065 | 94.663 | 46743.86 |
| s15850.1 | 625 | 25 | 8 | 66 | 1002 | 41550 | 0.066 | 95.694 | 59461.09 |
| s15850.1 | 625 | 25 | 10 | 66 | 1233 | 48975 | 0.078 | 97.706 | 83019.74 |
| s15850.1 | 625 | 25 | 11 | 81 | 1356 | 56175 | 0.089 | 98.709 | 96543.31 |
| s15850.1 | 625 | 25 | 12 | 88 | 1420 | 59700 | 0.095 | 99.733 | 114753.56 |
| s15850.1 | 625 | 25 | 13 | 95 | 1448 | 62325 | 0.099 | 100.000 | 127918.80 |
| b14.1 | 289 | 17 | 0 | 216 | 250 | 33626 | 0.045 | 85.838 | 7310.89 |
| b14.1 | 289 | 17 | 1 | 287 | 218 | 42738 | 0.057 | 86.705 | 16520.68 |
| b14.1 | 289 | 17 | 2 | 377 | 225 | 55097 | 0.074 | 87.892 | 25899.69 |
| b14.1 | 289 | 17 | 3 | 477 | 243 | 69003 | 0.093 | 88.984 | 36228.38 |
| b14.1 | 289 | 17 | 4 | 662 | 278 | 106012 | 0.142 | 90.216 | 49101.41 |
| b14.1 | 289 | 17 | 5 | 804 | 281 | 127789 | 0.172 | 91.122 | 64176.50 |
| b14.1 | 289 | 17 | 6 | 991 | 284 | 156451 | 0.210 | 92.200 | 77742.39 |
| b14.1 | 289 | 17 | 7 | 1148 | 309 | 180897 | 0.243 | 93.202 | 92378.23 |
| b14.1 | 289 | 17 | 8 | 1287 | 341 | 202708 | 0.272 | 94.152 | 109924.09 |
| b14.1 | 289 | 17 | 9 | 1453 | 346 | 228191 | 0.306 | 95.172 | 126674.31 |
| b14.1 | 289 | 17 | 10 | 1602 | 349 | 251039 | 0.337 | 96.212 | 148285.70 |
| b14.1 | 289 | 17 | 11 | 1793 | 366 | 280551 | 0.377 | 97.381 | 170142.53 |
| b14.1 | 289 | 17 | 12 | 1928 | 377 | 301393 | 0.405 | 98.318 | 190629.62 |
| b14.1 | 289 | 17 | 13 | 2057 | 398 | 321487 | 0.432 | 99.242 | 219983.75 |
| b14.1 | 289 | 17 | 14 | 2181 | 414 | 340731 | 0.457 | 100.000 | 240957.95 |
| s38417 | 1681 | 41 | 0 | 53 | 1393 | 81016 | 0.029 | 96.530 | 20061.56 |
| s38584 | 1521 | 39 | 0 | 36 | 577 | 36543 | 0.043 | 99.509 | 5836.27 |

ModelSim software is used for coding and simulating the existing and proposed architectures. ISE Design suite can also be used but the ModelSim software is more user friendly and it has an inbuilt simulation environment, which most of the ISE design suite older versions failed to provide. The code has been written in Verilog Description language and simulated successfully for obtaining the output.

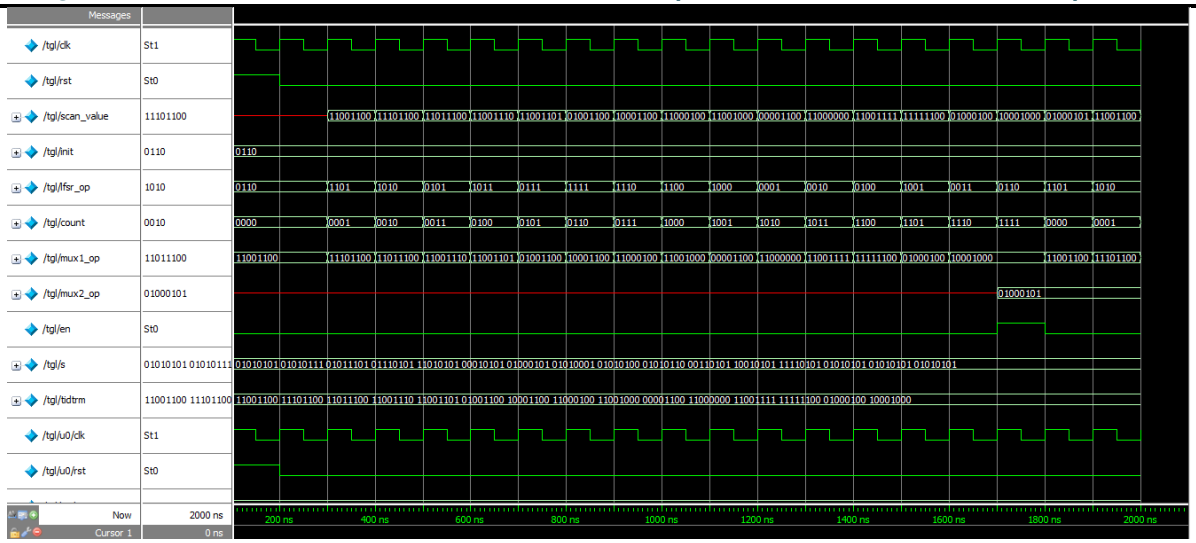



Fig 3. Simulated output of proposed BIST pattern generation

## VI . CONCLUSION

This project described a BIST approach that stores test data on- chip, and uses the stored test data to generate both random and deterministic tests on-chip. This approach offers a tradeoff between the amount of stored test data and the comprehensiveness of the test set that can be applied. The project explored this tradeoff in a specific context where the circuit under consideration has a large number of short scan chains, allowing storage of scan vectors. The initial stored test data is based on a stuck-at test set, but the set of target faults includes single-cycle gate-exhaustive faults. Under the approach described in the paper, the stored test data is enhanced gradually by test data from a gate-exhaustive test set, and the coverage of gate-exhaustive faults is increased gradually. Experimental results demonstrated this tradeoff for benchmark circuits.

## REFERENCES

[1] Irith Pomeranz, Fellow, IEEE, and Sudhakar M. Reddy, Fellow, IEEE “A Storage-Based Built-In Test Pattern Generation Method for Scan Circuits Based on Partitioning and Reduction of a Precomputed Test Set” , November 2022.

[2] Dong Xiang, Senior Member, IEEE, Mingjing Chen, and Hideo Fujiwara, Fellow, IEEE “Using Weighted Scan Enable Signals to Improve Test Effectiveness of Scan-Based BIST”, DECEMBER 2007.

[3] R.S Oliveria, J. semao, I C Teixeria, M B Santos, J P Teixeria, “ On-line BIST for Performance Failure Prediction under Aging Effects in Automotive Safety-Critical Applications.

[4] OSMAN EMIR EROL and SULE OZEV, Arizona State University, “Knowledge- and Simulation-Based Synthesis of Area-Efficient Passive Loop Filter Incremental Zoom-ADC for Built-In Self-Test Applications”.

[5] Knowledge- and Simulation-Based Synthesis of Area-Efficient Passive Loop Filter Incremental Zoom-ADC for Built-In Self-Test Applications

[6] S. Wang, K. J. Balakrishnan and S. T. Chakradhar, ”XWRC: Externally- Loaded Weighted Random Pattern Testing for Input Test Data Compres- sion”, Intl. Test Conf., 2005, Paper 24.2, pp. 1-10.

[7] E. J. McCluskey, ”Quality and Single-stuck Faults”, in Proc. Intl. Test Conf., 1993, p. 597.

[8] Y. Liu, N. Mukherjee, J. Rajski, S. M. Reddy and J. Tyszer, ”Deter- ministic Stellar BIST for In-System Automotive Test”, in Proc. Intl. Test Conf., 2018, pp. 1-9.

[9] S. U. Hussain, S. Yellapantula, M. Majzoobi and F. Koushanfar, ”BIST- PUF: Online, Hardware-Based Evaluation of Physically Unclonable Cir- cuit Identifiers”, in Proc. Intl. Conf. on Computer-Aided Design, 2014, pp. 162-169.

[10] A. H. Baba and S. Mitra, ”Testing for Transistor Aging”, in Proc. VLSI Test Symposium, 2009, pp. 215-220.