A Scalable BIST Architecture for Delay Faults

Martin Keim  Ilia Polian  Harry Hengster  Bernd Becker

Albert-Ludwigs-University, Institute of Computer Science
Am Flughafen 17, 79110 Freiburg im Breisgau, Germany
email: <keim, polian, hengster, becker>@informatik.uni-freiburg.de

Abstract

We present a scalable BIST (Built-In Self Test) architecture that provides a tunable trade-off between on-chip area demand and test execution time for delay fault testing. So, the architecture can meet test execution time requirements, area requirements, or any target in between.

Experiments show the scalability of our approach, e.g. that considerably shorter test execution time can be achieved by storing only a few additional input vectors of the BIST architecture. The gain of test execution time possible with the proposed method ranges from a factor of 2 up to a factor of more than 80000.

1 Introduction

Delay fault testing is likely to become industrially accepted in the near future. However, there is no single delay fault model, but several models that compete for acceptance. A discussion of the advantages and disadvantages of these fault models is published e.g. in [12]. Besides their differences, all these models have in common that a test for a fault consists of two successive test patterns (a test pair), that have to be applied to the circuit-under-test (CUT) at speed. The first pattern is denoted as the initialization vector. It is followed by the propagation vector.

Computing such two-pattern tests is known to be NP-hard [4]. However, there exist test pattern generators (TPG), e.g. for the path delay fault model [7]. Unfortunately, it becomes no longer manageable to apply all precomputed test patterns to the CUT by Automatic Test Equipment (ATE). Problems that come with this task are first of all the costs for a high speed ATE, that is necessary for today's chips. Furthermore, chips no longer consist of a single circuit, but host many different modules. In general, not all input pins of a module will be accessible. So, the precomputed test patterns cannot directly be applied to the target module.

BIST is an accepted method to solve the mentioned problems. Two approaches for delay fault BIST are possible:

One is to apply all possible transitions at the inputs of the CUT. This results in very long test execution time, since there are $2^n (2^n - 1)$ different pairs of n-input CUT (exhaustive testing). However, in [13] it was shown that it is sufficient to test robustly for path delay faults with test pairs that differ only at a single position (adjacency testing). The number of these patterns is $n \cdot 2^n$. BIST architectures that generate such patterns are presented in [15, 8, 5]. The test execution time of this approach is $n \cdot 2^n + 1$ for an n-input CUT with an area overhead of $3n$ measured in the number of memory elements used. Adding $n$ AND gates and considering so-called test cones, the actual test execution time can be reduced [8]. Accordingly, the number of memory elements is reduced to $2n$.

The other approach starts with a predetermined set of test pairs. It tries to build a hardware that generates sequences in which the test pairs are embedded, i.e. the initialization vector and the propagation vector are generated successively. E.g. in [6] the test pairs are embedded in the sequences generated by an LFSR, or in [16] in the sequences generated by a Multi Input Signature Register (MISR), shown in Figure 1. For an n-input CUT and k test pairs [16] needs $n$ memory elements for the MISR and $k \cdot 2^n$ clock cycles to generate all test pairs, with $k \leq k$, as explained later. Additionally, it needs to store $k' n$-bit input vectors (IVs) for the inputs of the MISR. The disadvantage of all these methods is, that the test execution time is fixed and raises exponentially with the number of inputs.

Based on [16], we present a method that offers a trade-off between the available clock cycle time for testing and the need of storing additional IVs for the MISR. The underlying idea is to store a very small set of IVs on chip or off chip and let the BIST hardware expand it so, that all given test pairs are embedded in the sequence generated.

It bases on the fact that the test pairs contain many don't-care values. This means, that many fully specified instances of each test pair exist. Some of them might be easier to embed in a sequence than others. Furthermore, we do not let the generator produce sequences for each IV that are of maximal length $2^n - 1$ but of specified length $g < 2^n - 1$. Then, at least one instance of each of the $k$ test pairs must occur in the first $g$ vectors of the sequence. For a given upper limit $k \cdot g < 2^n - 1$ of testing time, the proposed method computes a (minimum) number $k'$ of IVs needed to cover all given test pairs, resulting in a test execution time of $k' \cdot g$. Moreover, for a given limit $k' k$ of IVs, e.g. by an area restriction for storing the IVs, the proposed method will find a (minimum) number of clock cycles, such that all $k$ test pairs are covered by the generated test sequences.

Since the task of our method is to cover all given test pairs, it does not depend on a specific delay fault model nor on the CUT itself. Furthermore, we do not have any requirement on the given test patterns.

The proposed method runs as follows: Given a set of test pairs and a description of a MISR, find at first all input vectors of the MISR with at least one test pair is embedded in the full length sequence generated by the MISR. Then, the method tries to reduce the number of input vectors and/or the overall test execution time.

Comparing with other methods, the actual test execution time of the proposed method is reduced up to a factor of several magnitudes, e.g. for circuit s510 by a factor of $8.4 \cdot 10^3$ (compared to exhaustive testing), by a factor of $8.0 \cdot 10^3$ (compared to our reimplementation of [16]), and by a factor of $6.2 \cdot 10^3$ with respect to adjacency testing.

The rest of the paper is structured as follows: In the next section, we shortly introduce the MISR and some of its properties. Section 3 shows the proposed method. Experimental results are provided in Section 4 followed by the conclusions.

2 Multi Input Signature Register

We use a MISR as the generator structure for the test sequences, that are applied to the CUT. In Figure 1 a MISR of dimension $n$ is shown: It has $n$ inputs, $n$ outputs, and $n$ memory elements (stages) numbered from right to left. The Boolean values $b_0, \ldots, b_{n-1}$ de-
fine its feedback connections, i.e. $h_k = 1$ iff the $n - i - 1$th stage is selected.

The next state is a function of the current state and the IV, i.e., the vector $d$ applied to the inputs. Thus, the state transition relation $M$ can be defined as $y' = M(y, d)$, where $y'$ is the next state (vector), $y$ is the current state (vector), and $d$ is the input (vector):

$$y_i' = (M(y, d))i = \begin{cases} y_{i-1} \oplus d_i & 0 < i \leq n - 1 \\ \oplus_{i=0}^{n-1} y_{i-1} \oplus d_i & i = 0 \end{cases} \quad (1)$$

Given an input vector $d$ and a starting vector (seed) $y^0$, the MISR produces the sequence $M^1(y^0, d) = y^0, M^1(y^0, d) := M(y^0, d)$, $M^2(y^0, d) := M(M(y^0, d), d), \ldots$ on its outputs. We denote with $M^g(y^0, d)$ the whole sequence.

As for LFSRs the properties of a MISR can be described by a characteristic polynomial. If a characteristic polynomial is primitive, it has some interesting attributes. Since we take advantage of some of these in our work we choose the primitive polynomials from a list in [1]. An important property of a MISR of dimension $n$ is the following: given an IV $d$ and a starting state $y^0$, the sequence $M^g(y^0, d)$ has period $2^n - 1$. Furthermore: given two states $y$ and $y'$, there is one and only one IV $d$ with $y' = M(y, d)$.

The first property ensures, that essentially all output combinations are generated exactly once, if the MISR runs through all of its $2^n - 1$ states for each IV. In particular this means, if a fully specified test pair is embedded in a sequence $M^g(y^0, d)$ this pair occurs exactly once. But this does not mean that any test pair is embedded in all sequences (since there are $2^n - (2^n - 1)$ pairs, but the sequence length is only $2^n - 1$). However, the second property ensures, that for all test pairs there is (exactly one) sequence.

3 Solution Scheme

Since we let the MISR start with the seed $y^0 = (0, \ldots, 0)$ whenever possible (this seed may be forbidden due to a fully initialized test pair, that needs the IV $d \neq (0, \ldots, 0)$), we thereby obtain an ordering of the vectors generated by the MISR, i.e., we can talk of the $i$th vector produced by the MISR.

The proposed method takes advantage of the fact that, in general, the computed test pairs of a TPG contain a large number of don’t-care values. Thus, different fully specified instances of a test pair may need different IVs. Moreover, some instances of the test pair may be before others in the sequences, and it is very unlikely that all of them are at the end of the corresponding sequences. This means, it is not necessary to let the MISR run sequences of full length. In fact, the experimental results show, that nearly all of the test pairs are produced in very few steps.

Definition 1

1. A fully specified test pair $(v, w)$ is produced by the IV $d$ and the starting state $y^0$ iff there is an $i > 0$ with $v = M^g(y^0, d)$ and $w = M^g(y^0, d)$. We say $(v, w)$ is produced in $g$ steps iff $i \leq g$.

2. A fully specified test pair is produced in exactly $g$ steps iff it is produced in $g$ steps, but not in $g - 1$ steps.

3. A not fully specified test pair is produced (in $g$ steps) iff at least one of its fully specified instances is.

From now on, we do not mention $y^0$ explicitly and use produced by $d$ instead of produced by $d$ and $y^0$. It is important to note, that for a not fully specified test pair $(y, y')$ there are at least two different IVs that produce all instances of the test pair, as can be proven easily.

To solve the test-pair-embedding-task, we implemented two methods: Unbounded IV Minimization (UIM) and Bounded IV Minimization (BIM($g$)) for a limit $g$ of steps. Both methods run in two phases:

1. Determine for each test pair $TP$ the set $\Delta_i$ of all IVs
   (a) that produce it (Method UIM),
   (b) that produce it in $g$ steps (Method BIM($g$)).

2. Determine a subset $D$ of IVs so, that every test pair is produced by at least one IV from $D$. Since we want to minimize $|D|$, we are solving a set covering problem. This algorithm is the same for UIM and BIM($g$), and is discussed later in Section 3.2.

Figure 2 depicts the methods. Three test pairs ($TP_1, TP_2, TP_3$) are denoted by a triangle, a square, and a diamond. For the five IVs $d_1, \ldots, d_5$, the step in that an instance of a test pair is produced is shown horizontally. Three different limits $g_i$ are drawn. The figure shows that for UIM all test pairs can be produced with IV $d_k$. 
BIM($g_3$) needs only one IV more, namely $d_1$ and $d_2$. A different solution for BIM($g_3$) is $d_2$, $d_3$. For BIM with limit $g_2$ also IV $d_3$ is needed, since the test pair `diamond' is produced in $g_2$ steps neither by $d_3$ nor by $d_2$. Finally, the figure shows that for not all limits $g_2$ a solution exists. Here, $g_1$ is too small to produce all test pairs. (The `diamond' is not produced.)

3.1 Determine the IVs

3.1.1 IVs for UIM

Computing all IVs that produce all instances of a given test pair can easily be obtained by symbolic methods based on BDDs [3]. The necessary Boolean operations are executed efficiently on BDDs.

Firstly, we functionally describe a given MISR [16]: Let $M$ be the state transition relation of the MISR, then the characteristic function $\chi_{MISR} : \mathcal{E}^3 n \rightarrow \mathcal{E}$ of the state transition relation $M$ is defined as:

$$\chi_{MISR}(y', y, d) = 1 \iff M(y, d) = y' \quad (2)$$

$\chi_{MISR}$ can be constructed component-wise according to equation (1) [16]. $\chi_{MISR}(y', y, d)$ describes all possible transitions from state $y$ to the next state $y'$. But for the $j$th test pair $(V, W)$, we want to know only the set $\Delta_j$ of all IVs $d$ with $M(v, d) = w$ for all instances $v$ of $V$ and $w$ of $W$. For that restriction, the characteristic function $\chi_{\Delta_j} : \mathcal{E}^{2} \rightarrow \mathcal{E}$ (with $\chi_{\Delta_j}(d) = 1 \iff d \in \Delta_j$) can be obtained by restricting the $y'$ and $y$ variables in $\chi_{MISR}$ on $W$ and $V$, respectively:

$$\chi_{\Delta_j}(d) = \exists y'. y \chi_{MISR}(y', y | W, y | V, d) \quad (3)$$

In the set $\Delta_j$, all IVs producing any instance of the $j$th test pair $(V, W)$ are contained. For UIM the method continues with combining the $\Delta_j$, i.e. the subset $\mathcal{D}$ of IVs that covers all test pairs is determined. But for BIM($g$) the method continues with some simulations, as described next, before constructing $\mathcal{D}$.

3.1.2 IVs for BIM($g$)

For the IVs computed above there is no information available, which IV produces the test pair in how many steps. The only thing that is for sure is, that at least one instance of each of the $k$ test pairs occurs anywhere in the sequences $\{M^*(y^d, d), \forall d \in \Delta_j, j = \{1, \ldots, k\}\}$, resulting in a test application time of $k \cdot (2^n - 1)$.

In BIM($g$) only those IVs are of interest that produce the test pair in $g$ steps. We find these IVs by simulation. Two simulation algorithms have been implemented. The first one is based on a two-valued forward simulation, and the second one is using implicit backward simulation.

The two-valued forward simulation algorithm runs as follows: For the $j$th test pair, get the set $\Delta_j$ of IVs producing the test pair. Then, for each IV $d \in \Delta_j$ initialize the MISR with $y^d$. Continue to compute the next state until either the current and the next state is an instance of the test pair, or the limit $g$ is reached. In the latter case the test pair has not been produced in $g$ steps. The next state $M(y, d)$ of the MISR is implemented by bit operations on the machine word: At first, a left shift operation on $y$ is performed. Then, the bit representing $y_j$ is set to $\oplus_{j-1}^{\sqrt{n}} b$, where $b = y_j$. Finally, the XOR-operation with the IV $d$ is executed. Due to this implementation, the two-valued simulation runs very efficiently.

The algorithm of implicit backward simulation is shown in Figure 3. It works as follows: In the $j$th iteration, all IVs that produce the test pair in exactly $i$ steps are determined. To perform that, the IV-Current State relation $ICS_i : \mathcal{E}^{2n} \rightarrow \mathcal{E}$ is defined. $ICS_i$ represents the (IV, state) pairs producing the test pair in exactly $i$ steps. In $\chi_{\Delta_j, i}(d)$ all IVs are collected that produce the $j$th test pair in $i$ steps.

3.2 Set Covering

So far, we obtained for both, UIM and BIM($g$), for each test pair $TP_j$ the set $\Delta_j$ of all IVs producing it (in $g$ steps). Now, we want to combine and reduce the $\Delta_i$-th to a minimal size set of IVs $\mathcal{D}$.

Obviously, a set covering problem is to be solved. The matrix can be figured with $|\{TP_j\} \times |\mathcal{D}|$ rows, each $2^n$ columns (for the $2^n$ possible combinations of the n-bit IVs for $d$). An element of the matrix is marked, iff that test pair is produced (in $g$ steps) by the IV $d$. We are using a heuristic greedy approach to solve the covering problem. (Besides this heuristic, an exact algorithm has been implemented, also. Since the matrix to cover is very large an exact solution cannot be obtained for the considered circuits, except for the smallest one. Moreover, no different result has been obtained.)

For the two-valued simulation algorithm, a standard covering greedy heuristic has been used: In each iteration, take the IV (column) that has the most marked entries, i.e. that produces the most test pairs. Delete that column and the marked rows and start over until the matrix is empty. However, for large $n$ the matrix size of $k \cdot 2^n$ becomes unhandable. Thus, a symbolic set covering algorithm has been implemented. Furthermore, the number of test pairs to consider can then be reduced efficiently by row dominance.

3.2.1 Applying Row Dominance

The idea of row dominance (RD) is to exclude some rows that can be considered irrelevant, since they are covered by another row. Thus, they can be deleted without sacrificing any good solution: Imagine $\Delta_i$ and $\Delta_j$ with $\Delta_i \subseteq \Delta_j$. That is, every IV producing the $j$th test pair also produces the $i$th one. Thus, it makes no sense to choose a IV from $\Delta_j \setminus \Delta_i$ for producing the $i$th test pair, since we have to choose a IV from $\Delta_i$ anyway, which is also an element of $\Delta_i$ and thus produces the $i$th test pair, too. We say $\Delta_j$ is dominated by $\Delta_i$: If so, the row of the $j$th test pair can be deleted from the covering matrix. The dominance relationship can also be computed symbolically: $\Delta_i$ is dominated by $\Delta_j$ iff $\forall (\chi_{\Delta_i}) \lor \chi_{\Delta_j} = 1$. ($\lor$ is the OR operation).

3.2.2 Greedy Symbolic Set Covering

In the following, the greedy algorithm is described that symbolically solves the covering problem. The algorithm is similar to the one in [11] and has also been used in [16].
Algorithm: Implicit backward simulation
Input: jth test pair \( (V, W), g \in IV \)
Output: All IVs that produce \( (V, W) \) in \( g \) steps.

\[ ICS_s(y, d) := \exists y \chi_{MT \in S}(y \mid W, y \mid V, d) \]
\[ \chi_{\Delta_{j,v}}(d) := \exists y ICS_s(y \mid V, d) \]

for \( \text{int } i = 2; i \leq g; i++ \) do

\[ \text{tmp} := ICS_s(y, d) \]

\[ ICS_s(y, d) := \exists y \left( \text{tmp} \land \chi_{MT \in S}(y, y, d) \right) \]

\[ \chi_{\Delta_{i,v}}(d) := \exists y ICS_s(y \mid V, d) \lor \chi_{\Delta_{j,v}} \]

end

All IVs \( d \) with \( \chi_{\Delta_{i,v}}(d) = 1 \)

Figure 3: Algorithm of the implicit backward simulation.

Let \( m \) be the number of rows to consider. If RD has not been exploited, \( m \) is the number of test pairs \( k \) to produce, else \( m \) can be less. At first, the number \( j \) of test pair \( TP_j \) is encoded by the monom \( c_j \in B[^{\log_2 m}] \). Then, \( c_j \) is combined with the corresponding \( \Delta \):

\[ \chi_{\Delta_{j,v}}(c, d) = \bigvee_{j=1}^{m} (c_j \land \chi_{\Delta_j}) \] (4)

\( \chi_{\Delta_{j,v}}(c, d) \) describes the covering matrix. It can be represented symbolically by a single BDD.

Let the boolean variables used in the BDD be ordered as follows: The \( n \) variables \( d_1, \ldots, d_{n-1} \) representing the IVs \( d \) are ordered before the \( \log_2 m \) variables of the \( c_j \)-monoms. After the BDD has been built, a cut is placed between the \( c \) and \( d \) nodes of the BDD. The idea of the covering algorithm is as follows: A \( d \)-path \( c, c \) in the \( \chi_{\Delta_{j,v}}(c, d) \) consists of two parts: A \( d \)-component \( d \) in the \( d \) variables and a test component \( c \) in the \( c \) variables. A one-path \( d', c \) in the BDD means that the \( c \)th of the \( 2^n \) IVs produces the \( j \)th test pair. Since the On-sets of the BDDs rooted by \( c \) variables directly after the cut, i.e. an incoming edge of \( c \) crosses the cut, are (in general) larger than 1, any \( d \)-path leading to this root-nodes produces all test pairs in the On-set of that BDD:

- Calculate the size of the On-set for all BDDs that have a root-node directly after the cut.
- Find the maximal size of the On-set, and let the corresponding BDD be rooted by \( c_{\max} \).
- Select any path \( d' \) to \( c_{\max} \), i.e. an IV producing most test pairs.

After the IV has been chosen it is added to \( D \). Finally, the produced test pairs is removed from the matrix by symbolic operations.

After that, the sizes of the On-sets and a new \( c_{\max} \) must be recalculated. The algorithm stops if there are no more test pairs to produce, i.e. \( \chi_{\Delta_{j,v}} = 0 \). In this case \( D \) contains a number of IVs that produce at least one instance of every test pair.

3.2.3 Comparing the different Methods

If \( \text{BIM}(g) \) requires \( k' := |D_{\text{BIM}(g)}| \) IVs, the test application time is \( g k' \). Obviously, the smaller \( g \) is, the more IVs are tendentially needed. However, experimental results will show that \( k' \) is rising slower than \( g \) is falling, i.e. lowering \( g \) also lowers the total test execution time \( g k' \). However, there is no guarantee that for given \( g \) \( \text{BIM}(g) \) has a solution at all, as explained earlier.

On the other hand side UIM has in all cases a test execution time of \( k' \cdot (2^{n-1}) \), with \( k' := |D_{\text{UIM}}| \).

Figure 4: Test application comparison of different BIST schemes.

4 Experimental Results

We applied both methods UIM and \( \text{BIM}(g) \) to the combinational parts of ISCAS-89 [2] benchmark circuits. For generating two-patterns tests we used the tool TIP [9, 14], the successor of DYNAMITE [7] used in [16]. It computes robust test pairs for the path delay fault model. Note that due to TIP's advances, the numbers of test pairs here are smaller than the ones used in [16]. Therefore, we compare only to our reimplemention UIM of [16].

The computations were made on a Sun Ultra workstation with 512 MB main memory and 300 MHz. The computations for circuits with 23 or more inputs were made on a comparable workstation but with 1 GB main memory. All execution times are CPU times measured in seconds except for some values given in hours. Furthermore, we use a BDD package with improved techniques for the synthesis of the relational product operator that is fundamental for the image computation during state traversals [10].

The first table shows the results for the two-valued forward simulation and a standard covering heuristic. The second table shows the experimental results obtained by using the implicit backward simulation, row dominance, and symbolic covering. The first column of the tables contains the name of the circuit, followed by the number \#In of its inputs, and the number of test pairs, #TP. The next column gives execution time of the (optimal) adjacency
testing, i.e. \( \#\text{In} \cdot 2^{\#I_V} \).

The first column of the Results section describes at first the applied method. The column 'Left' contains the number of test pairs still to be considered by the set covering algorithm after RD. In Table 1, this column is omitted, since RD has not been applied. The next column is representing the amount of IVs needed to generate all test pairs, \( \#I_V \). The \( \#I_V \) vectors must be stored on the chip or in the ATE. Thus, that value describes the hardware demand of the applied method. The column 'Cover' contains the number of test pairs that have to be considered. However, the computing time for RD itself has to be considered. However, the computing time for RD itself mostly consumes the preserved seconds. This means, applying RD reduces the number of IV and has no negative impact on the overall execution time of the method.

Comparing the results in Table 1 with the results in Table 2 shows clearly, that for a small number of inputs, the two-valued based method is faster. However, if the number of inputs increases, the symbolic method outperforms the two-valued based method by RD since less rows have to be considered. However, the computing time for RD itself mostly consumes the preserved seconds. This means, applying RD reduces the number of IV and has no negative impact on the overall execution time of the method.

Comparing the results in Table 1 with the results in Table 2 shows clearly, that for a small number of inputs, the two-valued based method is faster. However, if the number of inputs increases, the symbolic method outperforms the two-valued based method, see s510. Furthermore, RD can only be applied efficiently in the symbolic case. Moreover, due to the size of the covering matrix, circuits with more than 25 inputs can hardly be processed. The symbolic method has no this restriction. Finally, the method UIM performs well only in the symbolic case.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#In</th>
<th>#TP</th>
<th>Name #In #TP</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIM(8)</td>
<td>37</td>
<td>296</td>
<td>708497</td>
</tr>
<tr>
<td>BIM(256)</td>
<td>16</td>
<td>4066</td>
<td>512.00</td>
</tr>
<tr>
<td>BIM(128)</td>
<td>17</td>
<td>2176</td>
<td>963.76</td>
</tr>
<tr>
<td>BIM(64)</td>
<td>20</td>
<td>1280</td>
<td>1638.40</td>
</tr>
<tr>
<td>BIM(32)</td>
<td>21</td>
<td>672</td>
<td>3120.76</td>
</tr>
<tr>
<td>BIM(16)</td>
<td>29</td>
<td>464</td>
<td>4519.72</td>
</tr>
<tr>
<td>BIM(8)</td>
<td>37</td>
<td>296</td>
<td>708497</td>
</tr>
<tr>
<td>BIM(4)</td>
<td>14</td>
<td>4</td>
<td>512.00</td>
</tr>
<tr>
<td>BIM(2)</td>
<td>7</td>
<td>4</td>
<td>256.00</td>
</tr>
<tr>
<td>BIM(1)</td>
<td>3</td>
<td>4</td>
<td>128.00</td>
</tr>
</tbody>
</table>

Table 1: Results for the two-valued based simulation.
The overall results for BIM(\(g\)) show the following: given BIM(\(g\)) requiring \(k^I\) IVs and BIM(\(2g\)) requiring \(k^I\) IVs, then \(k^I < 2 \cdot k^I\). That means, by lowering \(g\) the overall length of the sequence is lowered, too. So, a practical approach would be to divide the total number of clock cycles available by the number of test pairs. This is the time, each test pair may run, i.e. \(g\). However, since many test pairs are eliminated by RD the actual test application will run in less time than predicted. So, \(g\) may be increased in a second iteration to further reduce the number of IVs.

The overall results for UIM show that the number of IVs, #IVs, is usually not much less or even greater than the number of circuit's inputs, #In, which is the key number for adjacency testing. Since UIM results in \(2^{\#In} \cdot \#IVs\) test vectors generated by MISR, it is not significantly better or even worse than an adjacency testing scheme generating \(2^{\#In} \cdot \#In\) vectors. Since there is usually an upper bound for OvL, determined by the clock rate and maximal test execution time, some values are unacceptable. For example, when testing a 20 MHz device for at most 0.1 s, OvL must be \(\leq 2 \times 10^{06}\), which is not fulfilled by most UIM results. Furthermore, often there is a small \(g\), so that BIM(\(g\)) requires about the same amount of IVs as UIM, but with tremendously less test vectors, i.e. with tremendously shorter test execution time.

### 5 Conclusions

In this paper a scalable BIST architecture for delay faults has been presented. A MISR is used to generate sequences in which a given set of test pairs is embedded. These sequences have the property that at least one instance of all test pairs is contained within the first \(g\) vectors of the sequences. In many cases this results in a test execution time that is several magnitudes shorter than adjacency testing and UIM as well. Shorter sequences can be obtained in exchange for additional IVs of the MISR. Thus, the presented method offers a trade-off between area for storing the IVs and test execution time. A large number of experimental results show this scalability. Finally, if the IVs are not stored on the chip, but can be received from Automatic Test Equipment – at reasonable slower speed than delay fault testing would demand – the showed improved delay fault test execution time is nearly for free.

### References


